CHAPTER 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 Overview

The term “Feed” (or news-feed in case of social networks like Facebook) is a data representation format used for providing users or subscribers with frequently updated content. Generally, a feed is populated with content that is provided by publishers that are subscribed by a user. This data can be sorted based on the time of release or through other factors that help in acquiring the importance or relevance of the data to that user.

The steps involved at generating a unique feed for every user is decided based on the content that is served at the end. Several questions [4] need to be answered to decide the technique required such as:

- Are there more producers than consumers?
- What is the maximum acceptable latency?
- Will the feed items be sorted in the chronological order or ranked selectively?
- Will the feed be infinitely long or of fixed length?
- Whether or not ads are a part of the feed to monetize the feed service, etc.

After answering the above questions one needs to choose a mode for publishing the feed. They are primarily of two kinds [12]:

- “Push” Model/Fan-out-on-write
- “Pull” Model/or Fan-out-on-load

This project tries to implement a simple, efficient and scalable Hybrid Feed Generation algorithm that uses a combination of the above two publishing methods and show it’s working on a sample educational social networking platform called “QuickNotes” where users can share educational content.

1.2 History

The first Twitter prototype, developed by co-founders Dorsey and Florian Weber, was used as an internal service for employees Odeo and the full version was introduced publicly on July 15, 2006. At the beginning Twitter was considered as a combination of a social network as well as a microblogging service. Twitter was one of a kind service at the time that was capable of generating custom feeds to each user making it more unique and special to that user.

On September 6, 2006 the social network Facebook had introduced a “News Feed” feature similar to twitter’s. On 2018 Facebook announced significant changes to the News Feed, reconfiguring its algorithms to prioritize content from family members and friends over that of companies, outside media sites, or other public content. The shift is to improve the amount of meaningful content viewed.

On October 6, 2010 Instagram was founded, which grew up to become a very widely popular social networking platform. It’s platform was built using the latest HTML 5 web technology and was considered to be bleeding edge at the time. Instagram had used a more sophisticated news feed algorithm that was capable of giving more relevant content to its users than any other social network at the time. Another new development in terms of functionality in it’s newsfeed was when Instagram added an offline mode in April 2017, in which content previously loaded in the news feed was available without an Internet connection, and users could comment, like, save media, and unfollow users, all of which would take effect once the user goes back online.

CHAPTER 2
LITERATURE SURVEY

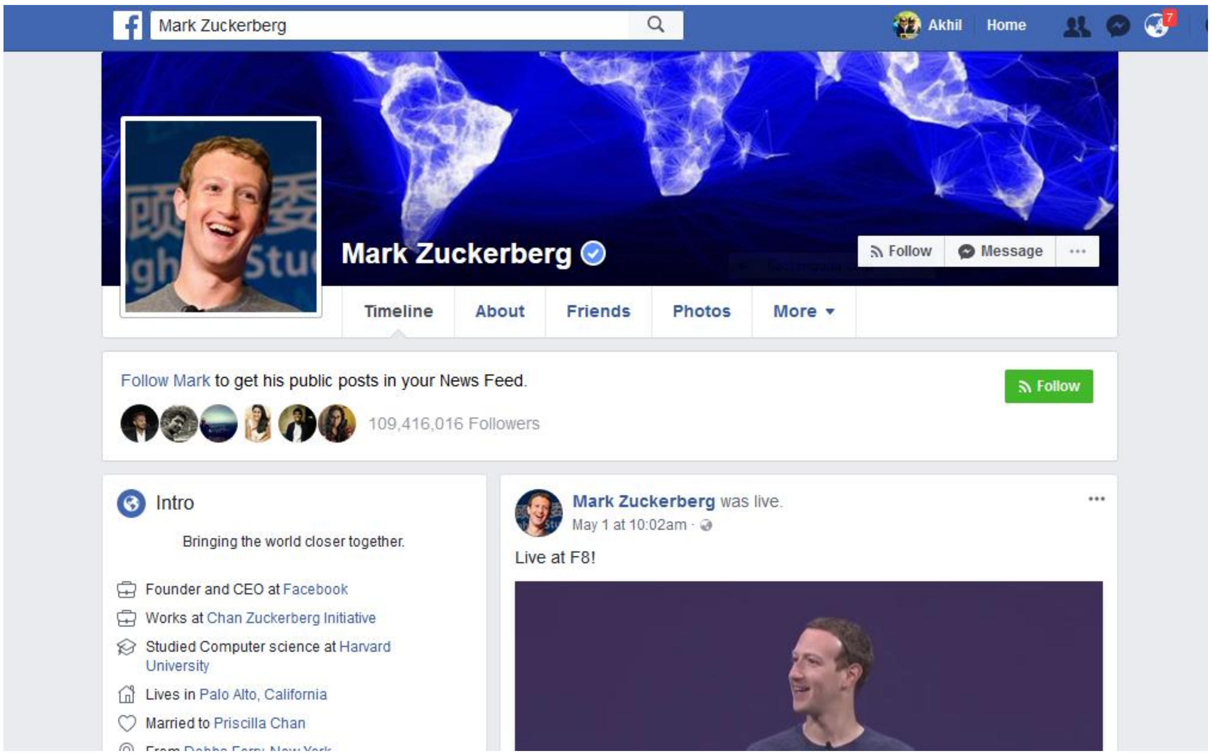
CHAPTER 2

LITERATURE SURVEY

In any social networking website there exists a feature called the news-feed. Generating this news feed is an expensive task since each user's feed is unique unlike a generic feed from regular news websites. Several social networks have implemented various tactics to tackle this problem. The various ways in which a feed is generated is a hot topic of research for quite some time now.

2.1 Activity Stream

An activity stream is a list of recent activities performed by an individual, typically on a single website [8]. For example, Facebook's News Feed is an activity stream. Activities in a stream include publishing a post, commenting on a post and 'liking'. We can see in Mark Zuckerberg's activity stream in figure 2.1.



Follow Mark to get his public posts in your News Feed.

109,416,016 Followers

Bringing the world closer together.

Founder and CEO at Facebook

Works at Chan Zuckerberg Initiative

Studied Computer science at Harvard University

Lives in Palo Alto, California

Married to Priscilla Chan

Live at F8!

Fig 2.1 An example of Facebook's activity stream

A typical scenario is when a user subscribes to a topic or follows a user. When an activity is generated, all users that are subscribed are notified - either immediately or eventually when the user logs in or refreshes the feed [14].

Feeds can be displayed in chronological occurrence of events or tailored for the user using a custom algorithm. If multiple events are combined over time it is called an aggregated feed [15].

2.2 Models

There are primarily two strategies for managing user feed events [13], they are explained below:

- Push Model / Fan-out-on-write (As shown in figure 2.2) - Each activity is pushed to a generated feed maintained for every consumer.

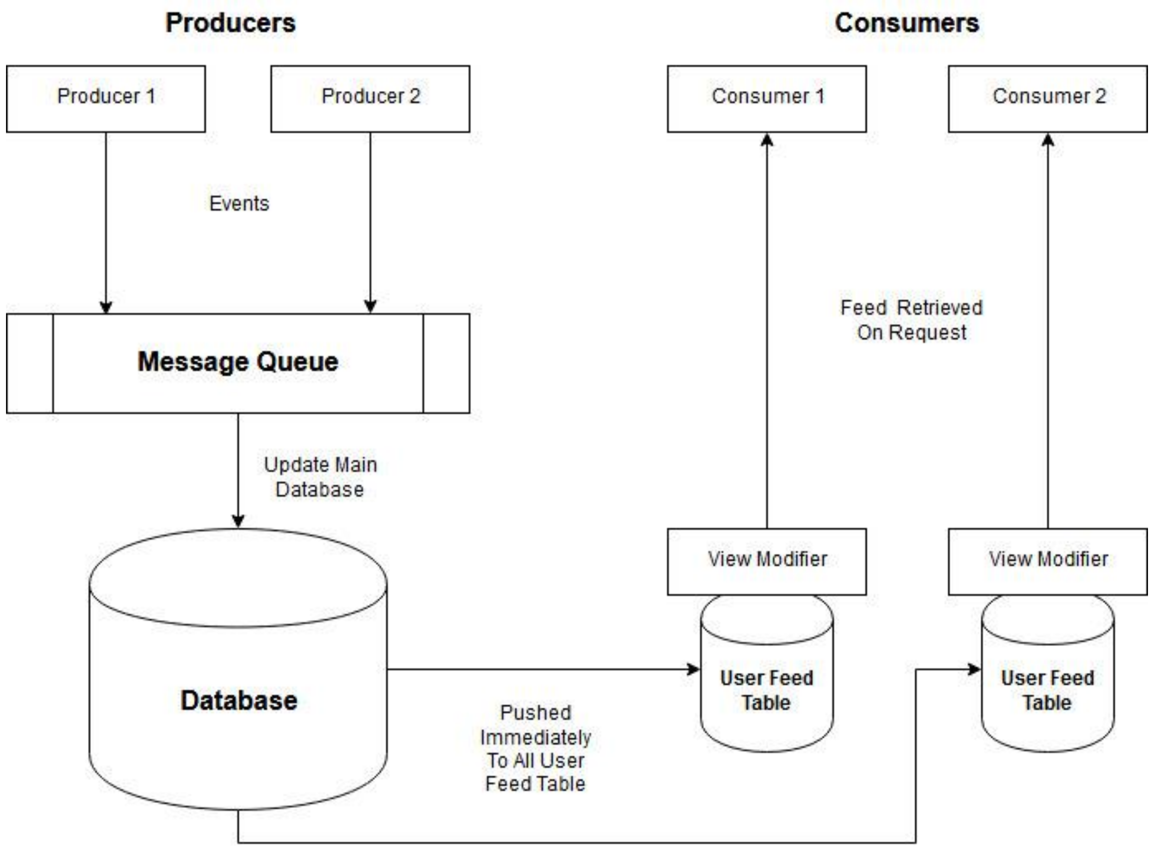


Fig. 2.2 Diagrammatic Representation of Push Model

- Pull Model / Fan-out-on-load (as shown in figure 2.3)- Activities are retrieved from producers when user logs in or refreshes the feed.

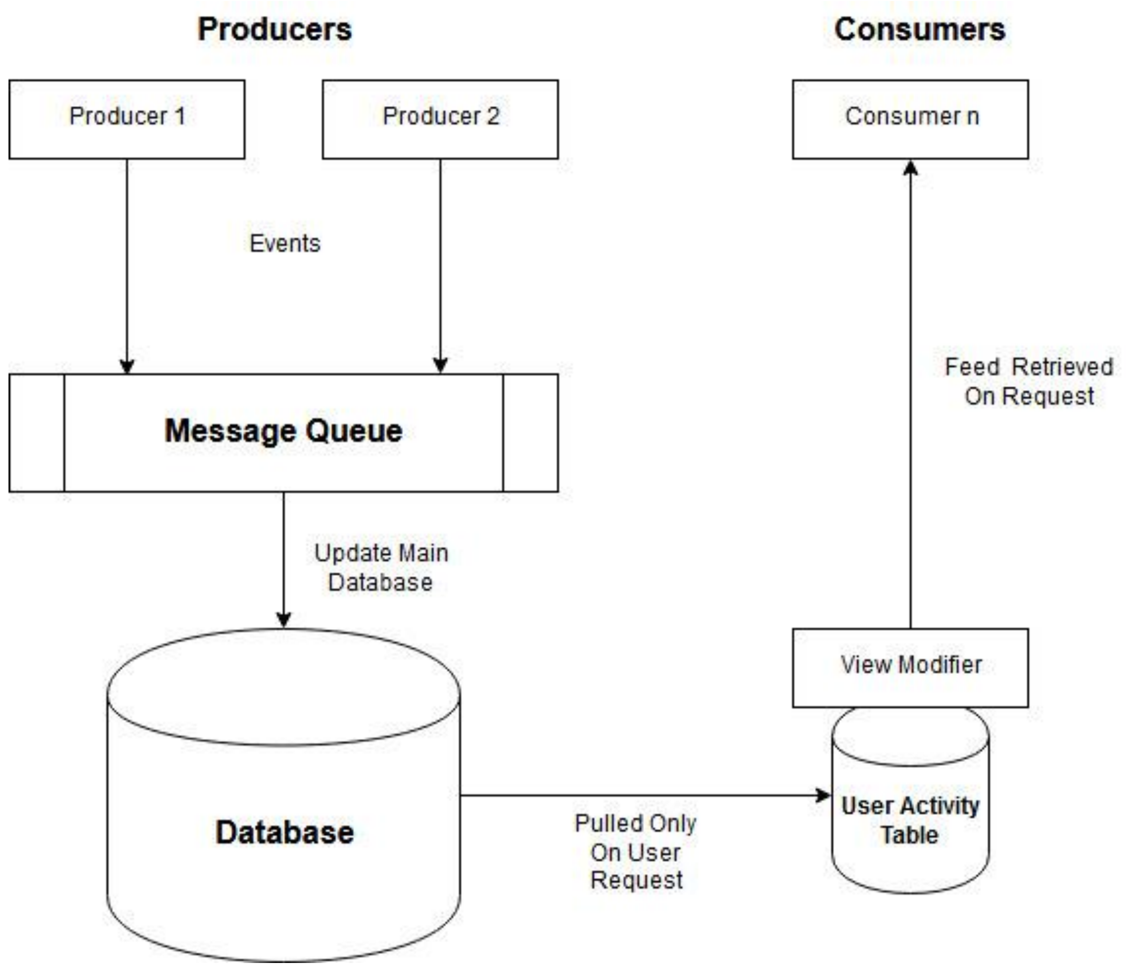


Fig. 2.3 Diagrammatic Representation of Pull Model

Using the push model results in lower latency because feeds are pre-generated. This involves a huge number of writes, but reads are really fast. This can lead to higher resource consumption and potentially wasteful if users log in infrequently. In contrast, using the pull model requires processing time to generate a feed but is much more resource friendly [7].

Many of the popular platforms today use a combination of both. Although this leads to better performance, it significantly increases complexity of the architecture.

2.3 Message Queue

In modern cloud architecture, applications are decoupled into smaller, independent building blocks that are easier to develop, deploy and maintain. Message queues provide communication and coordination for these distributed applications [9]. In applications requiring a feed, each item in the activity streams generated by users need to be processed. This can be done by pushing items to a message queue while another component performs actions on these such as ranking and pushing them to other users.

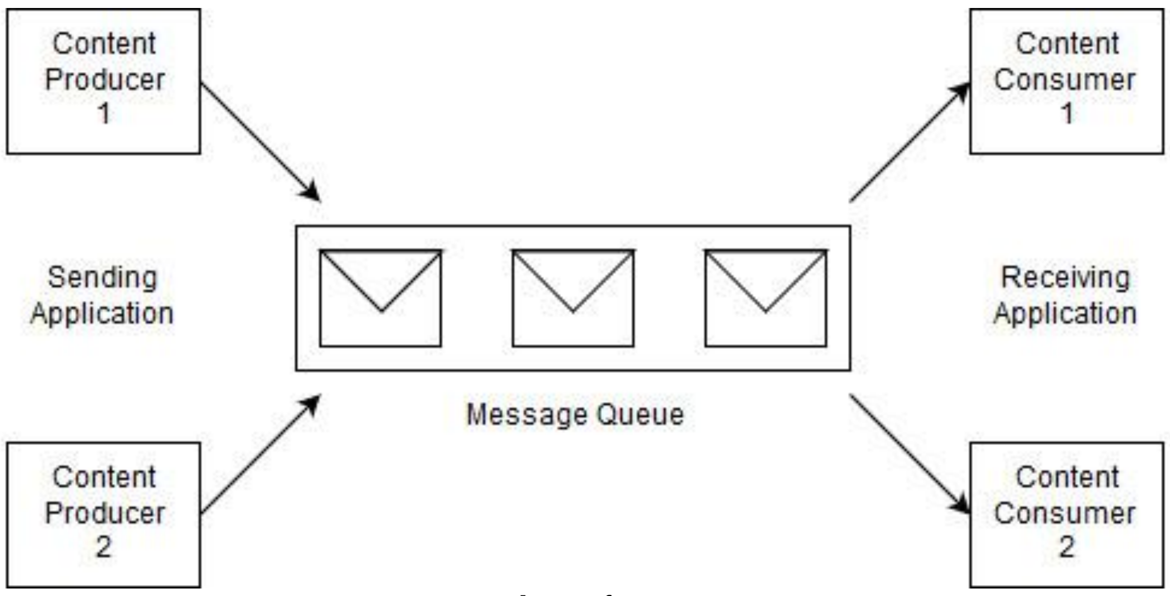


Fig. 2.4 Working of a message queue

There are a number of open source choices of messaging middleware systems, including Apache ActiveMQ, Apache Kafka, Apache Qpid, Beanstalkd, HTTPSQS JBoss Messaging, JORAM, RabbitMQ, Sun Open Message Queue, and Tarantool.

2.4 Facebook

Facebook's architecture uses a model that is primarily pull based [10]. News feeds are fetched using aggregators, which are query engines that accept user queries and retrieve items from the backend while performing aggregation, ranking and filtering. Aggregators use 'leaf' servers, which is a distributed storage layer that indexes most recent activity for each user.

2.5 Instagram

In contrast to Facebook, Instagram uses a predominantly push based model for generating a user's feed. When a user uploads a post, the activity is asynchronously pushed to each of the user's followers using a task manager and a message broker [11]. Cassandra, Redis, RabbitMQ, PostgreSQL and Memcached are used to manage data [3].

2.6 Twitter

Twitter uses a combination of both, pushing only to users that are currently active. Fanouts are implemented as microservices [5]. Cache is critical for Twitter and protects backing stores from heavy read traffic. It uses Manhattan, Blobstore, FlockDB and Redis among others [1].

2.7 Yahoo

Yahoo describes architectures and techniques for large scale applications that require the generation of activity feeds in its paper "Feeding Frenzy: Selectively Materializing Users' Event Feeds" [6]. It suggests selecting a push or pull action for each producer consumer pair based on their relative producing and querying frequencies and the cost of each action.

2.8 Pinterest

An implementation of the feed at Pinterest needs to be continually updated as users follow or unfollow other users and boards. New content is pushed out to the feeds of all followers. MySQL is used for persistence and HBase for backend storage [2].

CHAPTER 3

OBJECTIVE & SCOPE OF THIS PROJECT

CHAPTER 3

OBJECTIVE & SCOPE OF THIS PROJECT

3.1 OBJECTIVE

- To develop a hybrid user specific feed fetch model to improve the performance and scalability in social applications.
- Develop a social networking platform application to share academic resources among students and teachers (Called as QuickNotes).
- Implement the hybrid feed fetch model into this social platform to demonstrate performance enhancement.

3.2 SCOPE

In this project, we focus on addressing the following components of the feed generation model:

- Defining the feed generation problem formally as a partial view materialization problem.
- An analysis of the optimization problem that determines the events that needs to be pushed and the events that needs to be pulled. This describes the hybrid model that we propose, rather than using complete push or complete pull strategy, in order to minimize system load and maximize scalability while still providing latency guarantees.
- Algorithms for selectively pushing/pulling events to a consumer feed through decisions made locally on a per-producer/consumer basis and establishing the global performance benefit that it provides.
- An implementation of our techniques in an educational social networking platform to show that it performs well in practice.

Following are the components that are out of the scope of this project:

- Aggregation of events in the consumer feed.
- Revoking of events later from the feed.
- Changing position of the events in the consumer feed to avoid duplication (in case where multiple producers produce the same event).
- Diversity of the events in the feed.

CHAPTER 4

SYSTEM REQUIREMENTS & SPECIFICATIONS

CHAPTER 4

SYSTEM REQUIREMENTS SPECIFICATON

A Software Requirements Specification (SRS) is a document that describes the nature of a project, software or application. In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project/application. This document is also known by the names SRS report, software document. A software document is primarily prepared for a project, software or any kind of application.

4.1 FUNCTIONAL REQUIREMENTS

The functional requirements enlist the operations and activities that a system must be able to perform. Our project is incorporating our hybrid feed generation algorithm into an educational social networking platform called “QuickNotes”. The functional requirements are as follows:

1) Build a hybrid feed generating algorithm for the social network.
2) Users can register to the social networking site and login whenever he/she wishes to do so.
3) Users can create his/her profile, add, edit information present in various profile sections
4) Users should be able to choose whom to follow for content.
5) Users should be able to do social actions such as “upvote”, “share”, “comment” and “download” content.

4.2 NON-FUNCTIONAL REQUIREMENTS

The modelling, automatic implementation and runtime verification of constraints in component-based applications. Constraints have been assuming an ever more relevant role in modelling distributed systems as long as business rules implementation, design-by-contract practice, and fault-tolerance requirements are concerned. Nevertheless, component developers are not sufficiently supported by existing tools to model and implement such features, we propose a methodology and a set of tools that enable developers both to model component constraints and to generate automatically component skeletons that already implement such constraints. The methodology has been extended to support implementation even in case of legacy components.

4.3 OTHER NON-FUNCTIONAL REQUIREMENTS

4.3.1 Safety and Security Requirements

This requirement is important when it comes to a social networking platform where privacy and security of personal information is of utmost priority. Many software-intensive systems have significant safety and security ramifications and need to have their associated safety- and security-related requirements properly engineered. For example, it has been observed by several consultants, researchers, and authors that inadequate requirements are a major cause of accidents involving software intensive systems. Yet in practice, there is very little interaction between the requirements, safety and security disciplines and little collaboration between their respective communities. Most requirements engineers know little about safety and security engineering, and most safety and security engineers know little about requirements engineering. Also, safety and security engineering typically concentrates on architectures and designs rather than requirements because hazard and threat analysis typically depend on the identification of vulnerable hardware and software components, the exploitation of which can cause accidents and enable successful attacks.

4.3.2 Software Quality Attributes

The following factors are used to measure software development quality. Each attribute can be used to measure the product performance. These attributes can be used for Quality assurance(QA) as well as Quality control(QC). QA activities are oriented towards prevention of introduction of defects and QC activities are aimed at detecting defects in products and services.

- **Reliability**

  Measure if product is reliable enough to sustain in any condition give consistently correct results. Product reliability is measured in terms of working of project under different working environment and different conditions.

- **Maintainability**

  Different versions of the product should be easy to maintain. For development it should be easy to add code to existing system, should be easy to upgrade for new features and new technologies time to time. Maintenance should be cost effective and easy. System be easy to maintain and correcting defects or making a change in the software

- **Usability**

This can be measured in terms of ease of use. Application should be user friendly. The system must be Easy to use for input preparation, operation, and interpretation of output, and provide consistent user interface standards or conventions with our other frequently used systems. They should be easy for new or infrequent users to learn to use the system.

- **Portability**

This can be measured in terms of Costing issues related to porting, Technical issues related to porting, Behavioural issues related to porting. The social network developed as a result of this project must be able to work on multiple platforms, making portability an important requirement.

- **Correctness**

Application should be correct in terms of its functionality, calculations used internally and the navigation should be correct. This means application should adhere to functional requirements.

- **Efficiency**

To Major system quality attribute. Measured in terms of time required to complete any task given to the system. For example, system should utilize processor capacity, disk space and memory efficiently. If system is using all the available resources, then user will get degraded performance failing the system for efficiency. If system is not efficient then it cannot be used in real time applications.

4.4 SYSTEM REQUIREMENTS FOR THE PROJECT

OPERATING SYSTEM	:	Linux/Windows/MacOS/Android/IOS
PROCESSOR	:	X86/ARM Based Processor
RAM	:	2GB
OTHER REQUIREMENTS	:	HTML5 Enabled Browser

4.5 DEVELOPMENT REQUIREMENTS

4.5.1 Hardware Requirements

PROCESSOR	:	Intel i3 / Ryzen 3
RAM	:	2GB
MONITOR	:	15”
HARD DISK	:	20GB
KEYBOARD	:	Standard 102 Keys

4.5.2 Software Requirements

BACK-END	:	Django
API	:	Django Rest Framework
FRONT-END	:	ReactJS
DATABASE	:	MySQL
CLOUD PLATFORM	:	Azure

CHAPTER 5

SYSTEM ANALYSIS & DESIGN

CHAPTER 5

SYSTEM ANALYSIS & DESIGN

5.1 OVERVIEW OF THE PLATFORM (QuickNotes)

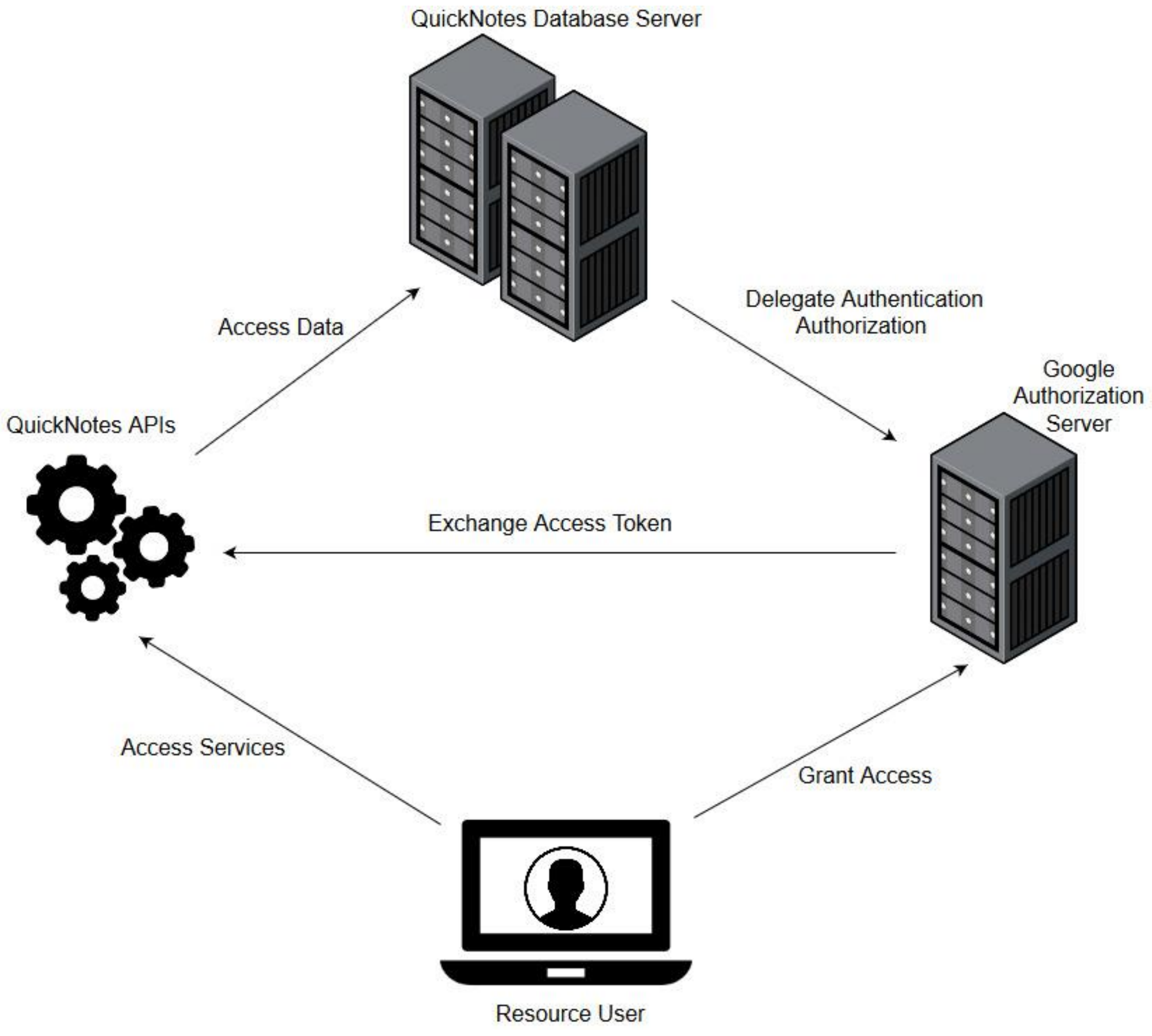

QuickNotes Database Server
Access Data
Delegate Authentication Authorization
QuickNotes APIs
Google Authorization Server
Exchange Access Token
Access Services
Grant Access
Resource User

Fig. 5.1 Overview of QuickNotes

When a user wants to login to the service he/she would have to use Google's OAuth service to authenticate themselves. After this is done the app uses the QuickNotes Django APIs to access data from the database servers. The QuickNotes backend server would delegate the authentication authorization of the user to the Google's authorization server. There also exists APIs to generate a news feed for a user based on the content in the database.

5.2 MODULES OF QUICKNOTES

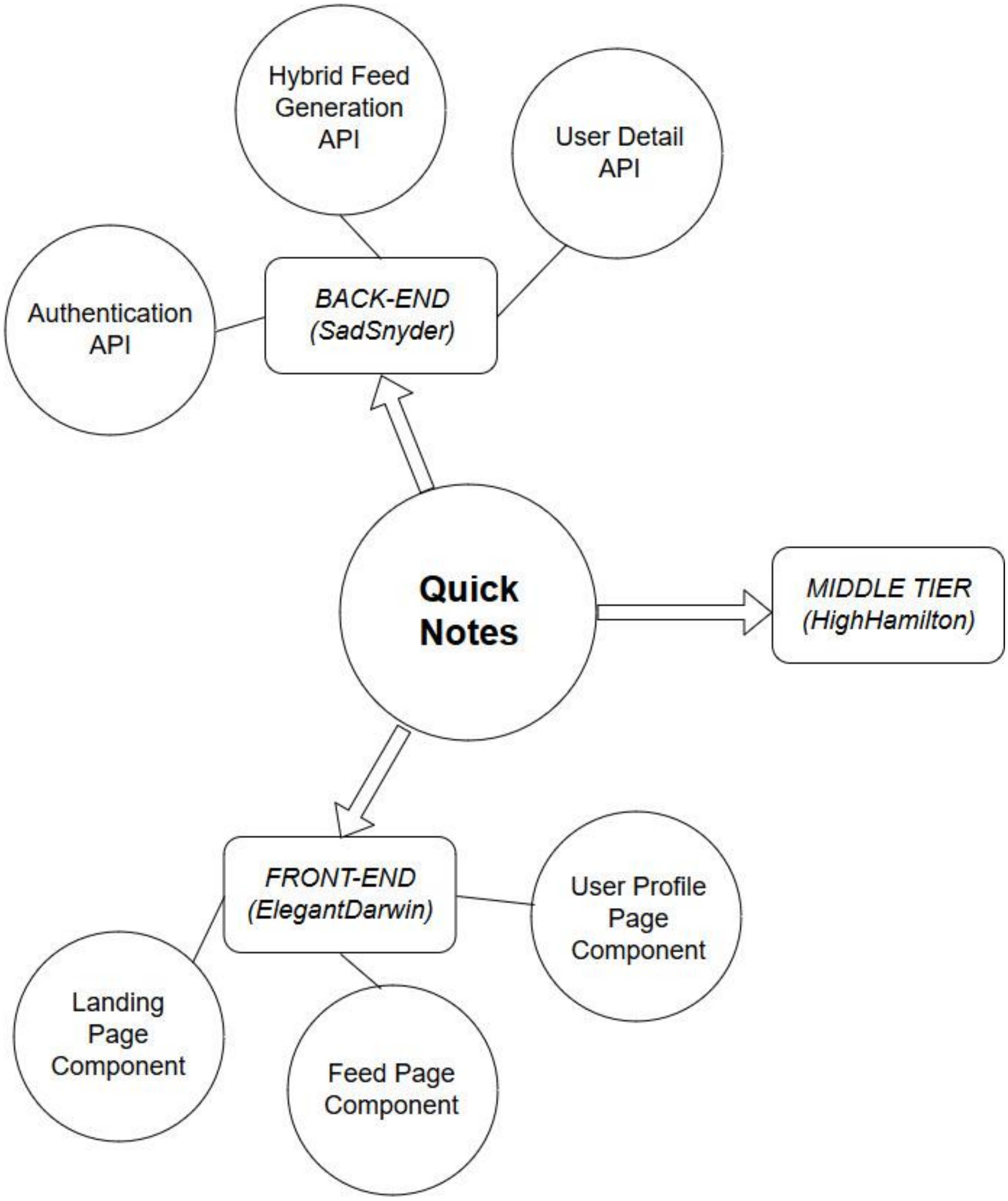


Fig. 5.2 QuickNotes Modules

The QuickNotes platform is divided primarily into three modules(Fig 6); they are:

(1). Back-End (codename: *SadSnyder*): This consists of the several APIs that are developed using Django. These APIs take care of user authentication, user feed generation and providing user details.

(2). Front-End (codename: *ElegantDarwin*): This consists of ReactJS components that can dynamically interact with each other. This consists of several components for various aspects of the front end such as the landing page, the news-feed page and the user details page.

(3). Middle Tier (codename: *HighHamilton*): This module has the API interactions that are responsible for communication with *SadSnyder* and drives the code of *ElegantDarwin* to give a very dynamic interaction.

Sub-Components of SadSnyder:

- Authentication API: This API uses Google's OAuth to authenticate the users to the service
- Hybrid Feed Generation API: This API is responsible for generating a custom news feed for the user. This API is a combination of PUSH and PULL feed generation methods.
- User Details API: This API returns the details of the user such as username, date of birth, user social counts etc.

Sub-Components of ElegantDarwin:

- Landing Page component: This page is the first thing that a user would see before logging in.
- Feed Page Component: This page displays the user's custom news feed. This component through *HighHamilton* calls the Hybrid Feed Generation API
- User Profile Page Component: This page shows the details of the user. This component calls the User Details API through *HighHamilton.*

5.3 USE-CASE DIAGRAM

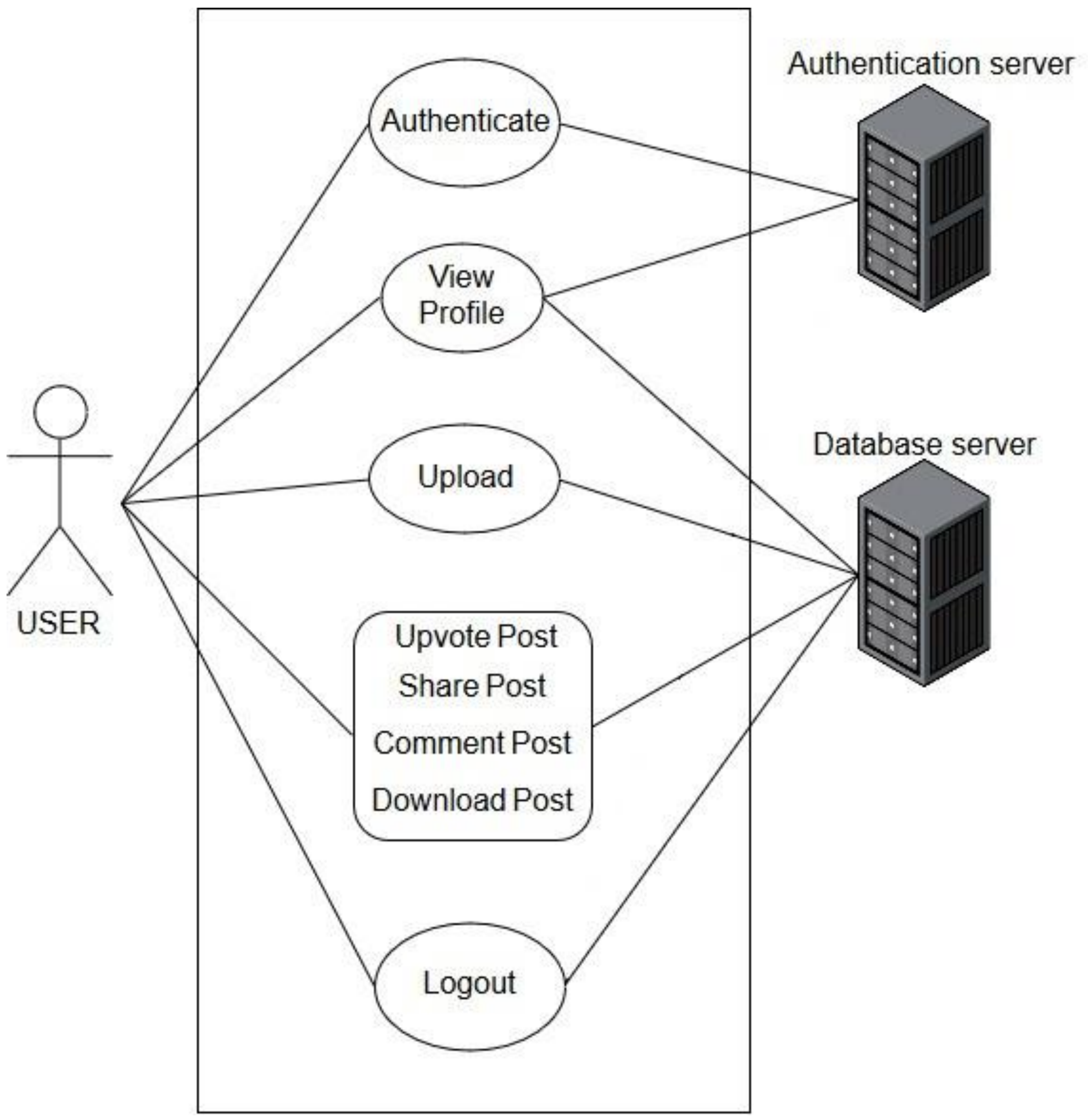


Fig. 5.3 Use-case diagram

5.4 DATABASE SCHEMA

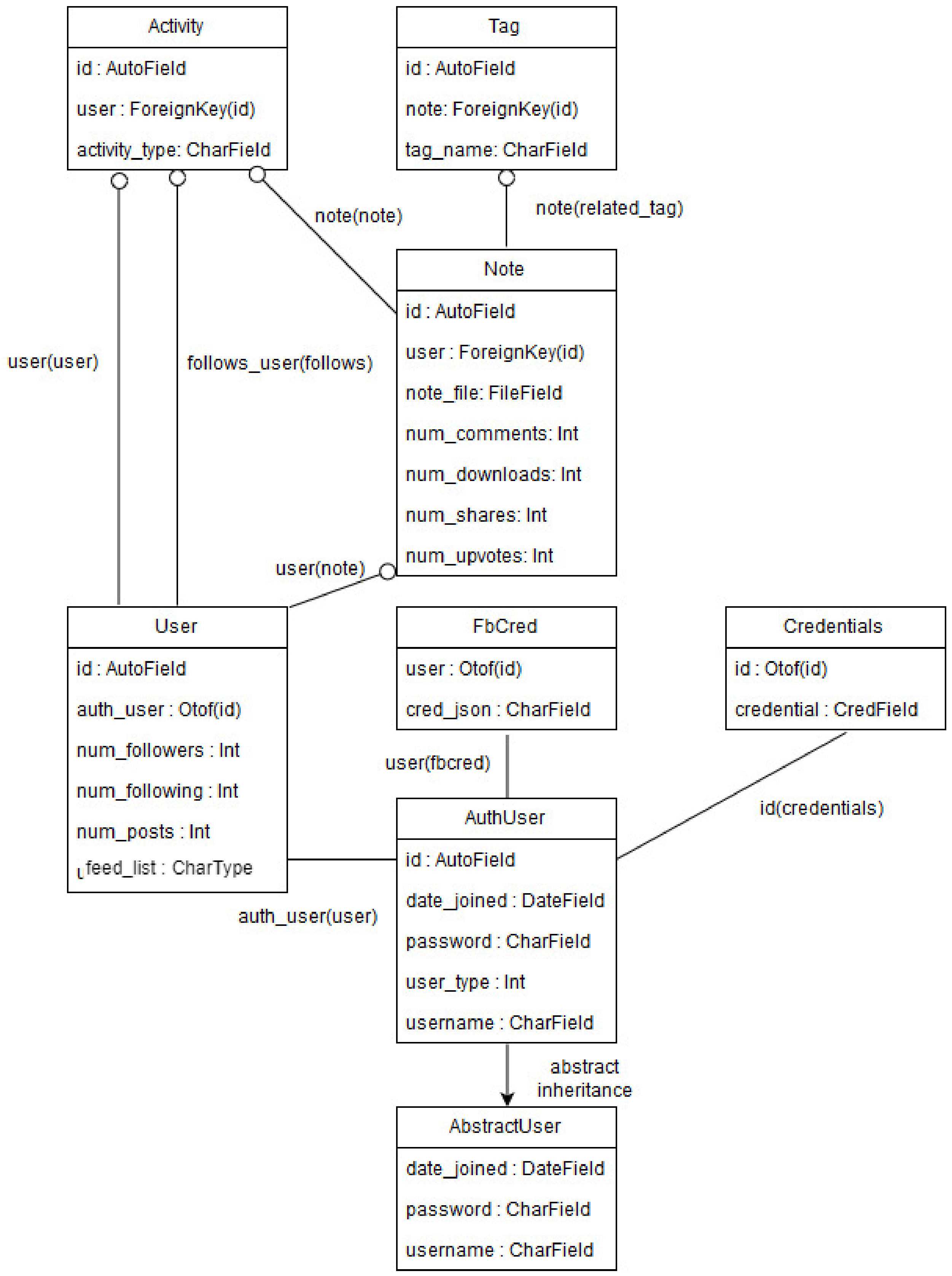


Fig. 5.4 Database schema

5.5 DATABASE TABLE DESCRIPTION

- **AuthUser:** AuthUser tables are created when a user registers into the platform using OAuth. This table holds User login data such as password and username. This table entry is a must for the other tables to exist for a user.
- **User:** User tables are created when a user logs into the platform. This table holds User profile data such as number of current followers, number of people following that user, total number of posts made by this user and also the users feed list.
- **FbCred:** This table contains the Facebook profile details of the user such as profile photo, full name, age, etc when he/she uses Facebook's OAuth to log into the platform.
- **Credentials:** This table contains the Google profile details of the user such as profile photo, full name, age, etc when he/she uses Google's OAuth to log into the platform.
- **Note:** Whenever a user posts a note onto the platform, the details of the note is saved in this table. Details such as the attached file, the number of comments on the post, the number of downloads, shares and upvotes along with a unique id for that post are available in a record from this table
- **Activity:** This table keeps track of what a user has done with a post. It says if the post is shared, commented or upvoted. It relates the User with Note records through these actions.
- **Tag:** This table contains all the possible tags that any content would belong to. A note can have multiple tags associated with it. This table helps one segregate content in categories.

CHAPTER 6

IMPLEMENTATION & PROPOSED MODEL

CHAPTER 6

IMPLEMENTATION & PROPOSED MODEL

6.1 LANGUAGES USED

6.1.1 Python (Back-End)

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure. This feature is also sometimes termed the off-side rule.

6.1.2 Features of Python

- **Easy-to-learn**: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read**: Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain**: Python's source code is fairly easy-to-maintain.
- **A broad standard library**: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode**: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable**: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable**: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases**: Python provides interfaces to all major commercial databases.
- **GUI Programming**: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable**: Python provides a better structure and support for large programs than shell scripting.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.

6.1.3 JavaScript ES6 Syntax (Front-End)

JavaScript ECMAScript (ES) is a scripting language specification standardized by ECMAScript International. It is used by applications to enable client-side scripting. Languages like JavaScript, Jscript and ActionScript are governed by this specification.

JavaScript was developed by Brendan Eich, a developer at Netscape Communications Corporation, in 1995.JavaScript started life with the name Mocha, and was briefly named LiveScript before being officially renamed to JavaScript. It is a scripting language that is executed by the browser, i.e. on the client’s end. It is used in conjunction with HTML to develop responsive webpages.

JavaScript can run on any browser, any host, and any OS. Node.js is an open source, cross-platform runtime environment for server-side JavaScript. Node.js is required to run JavaScript without a browser support. It uses Google V8 JavaScript engine to execute the code.

ES6 ignores spaces, tabs, and newlines that appear in programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

JavaScript is case-sensitive. This means that JavaScript differentiates between the uppercase and the lowercase characters. Each line of instruction called a statement need not be separated using a semi colon. Semicolons are optional in JavaScript.

6.1.4 ES6 Syntax Features

- Support for constants like traditional languages
- Block Scope
- Support for Arrow Functions
- Extended Parameter Handling
- Template Literals
- Extended Literals
- Enhanced Object Properties
- De-structuring Assignment
- Modules
- Classes are supported
- Availability of Iterators
- Generators
- Java styled Collections support
- New built in methods for various classes

6.2 FRAMEWORKS USED

6.2.1 Django (Back-End)

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

6.2.2 Django Advantages

- **Object-Relational Mapping (ORM) Support** − Django provides a bridge between the data model and the database engine, and supports a large set of database systems including MySQL, Oracle, Postgres, etc. Django also supports NoSQL database through Django-nonrel fork. For now, the only NoSQL databases supported are MongoDB and google app engine.
- **Multilingual Support** − Django supports multilingual websites through its built-in internationalization system. So you can develop your website, which would support multiple languages.
- **Framework Support** − Django has built-in support for Ajax, RSS, Caching and various other frameworks.
- **Administration GUI** − Django provides a nice ready-to-use user interface for administrative activities.
- **Development Environment** − Django comes with a lightweight web server to facilitate end-to-end application development and testing.

6.2.3 ReactJS (Front-End)

React is a front-end library developed by Facebook. It is used for handling the view layer for web and mobile apps. ReactJS allows us to create reusable UI components. It is

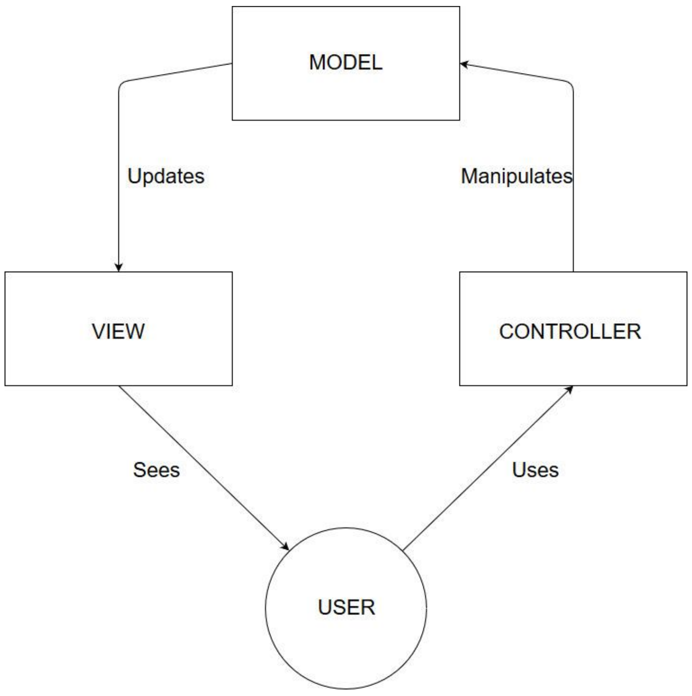


Fig. 6.1 MVC architecture

currently one of the most popular JavaScript libraries and has a strong foundation and large community behind it.

React is a library for building composable user interfaces. It encourages the creation of reusable UI components, which present data that changes over time. Lots of people use React as the V in MVC (Figure 6.1). React abstracts away the DOM from you, offering a simpler programming model and better performance. React can also render on the server using Node, and it can power native apps using React Native. React implements one-way reactive data flow, which reduces the boilerplate and is easier to reason about than traditional data binding.

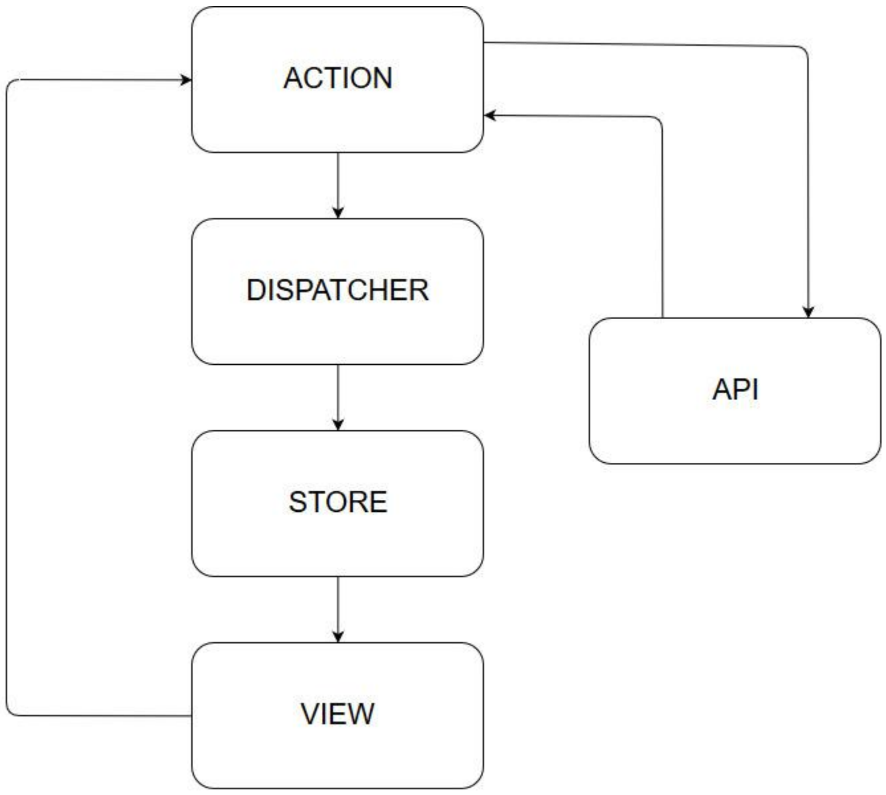


Fig. 6.2 ReactJS flux process

6.2.4 ReactJS Advantages

- **JSX** − JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
- **Components** − React is all about components. You need to think of everything as a component. This will help you maintain the code when working on larger scale projects.

- **Unidirectional data flow and Flux** − React implements one-way data flow which makes it easy to reason about your app. Flux is a pattern that helps keeping your data unidirectional (Figure 6.2 shows the ReactJS Flux Process).
- **Uses virtual DOM** which is a JavaScript object. This will improve apps performance, since JavaScript virtual DOM is faster than the regular DOM.
- Can be used on client and server side as well as with other frameworks.
- Component and data patterns improve readability, which helps to maintain larger apps.

6.3 MICROSOFT VISUAL STUDIO CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences. It is free and open-source, although the official download is under a proprietary license.

Visual Studio Code is based on Electron, a framework which is used to deploy Node.js applications for the desktop running on the Blink layout engine. It supports a number of programming languages and a set of features that may or may not be available for a given language, as shown in the following table. Many of Visual Studio Code features are not exposed through menus or the user interface. Rather, they are accessed via the command palette or via a .json file (e.g., user preferences). The command palette is a command-line interface.

6.4 MYSQL

Structured Query Language (SQL) is a domain-specific language used in programming and designed for managing data held in a relational database management system (In this case MySQL).

Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language, data manipulation language, and data control language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements.

SQL was one of the first commercial languages for Edgar F. Codd's relational model, as described in his influential 1970 paper, "A Relational Model of Data for Large Shared Data

Banks." Despite not entirely adhering to the relational model as described by Codd, it became the most widely used database language.

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. Since then, the standard has been revised to include a larger set of features. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments.

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

6.5 THE PROPOSED MODEL

According to Yahoo's research paper [1], the optimal system utilisation can be achieved by making local push/pull decisions on a per producer-consumer pair basis, depending on the relative frequencies of publishing and querying. Our project suggests a less complex alternative by associating the push/pull decision with each user.

Here, individual users are classified as either push-target or pull-target based on the frequency of querying for feed. If a user queries for updates frequently, she's classified as a push-target. If a user queries relatively less, she's a pull-target. The threshold frequency to decide whether a user is push or pull target can be decided based on the average query frequency of all users and current system performance.

If a user has query frequency below the threshold, she is a pull-target; else push. When an event is generated, it is only pushed to all the followers of the producers that are push-targets. Thus when a user requests for a feed, the system first checks to see what kind of user it is. In case of a push-target, the feed is directly fetched from that user's materialized feed table. In the other case, the feed is generated by fetching the activity of the producers in that user's network. This is shown in figure 6.3.

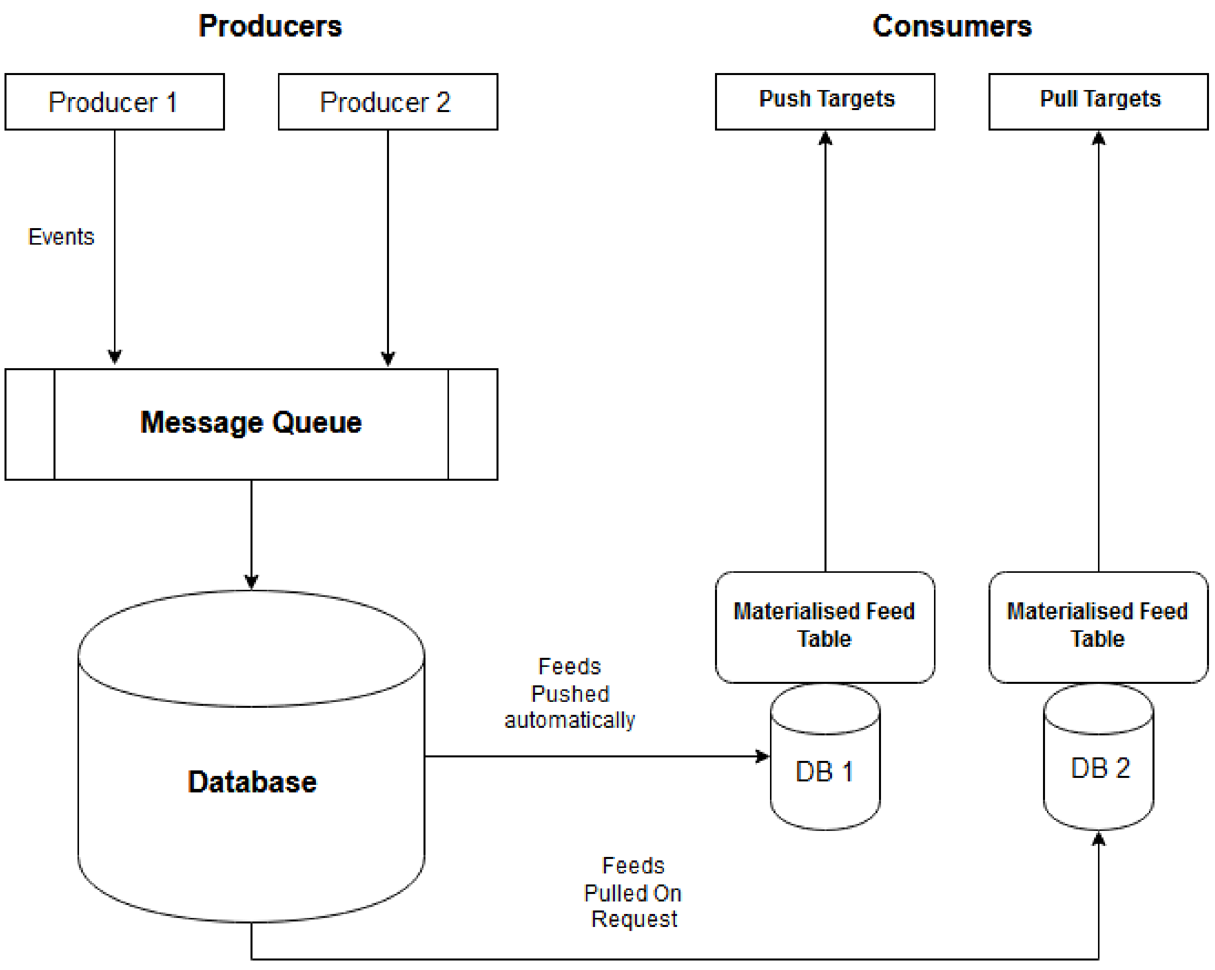


Fig. 6.3 Diagrammatic representation of hybrid model

6.6 THE DESIGNED ALGORITHMS

The abstract implementation of the push model of feed generation is show in algorithm 1.

Algorithm 1 Algorithm for Push

Input: *e (Event), User*

Output: *User Feed*

```
1:  onEvent(e, User):
2:      for each follower of User:
3:          follower.Feed.push(e)
4:
5:  fetchFeed(User):
6:      return User.Feed
```

Each event by a producer is pushed to the feed table of all the subscribers of the producer. When a feed is requested by a user, it is fetched from the per-consumer materialized feed table. Ranking of items in the feed can be done either when the feed is fetched, or when an event is being pushed.

The abstract implementation of the pull model of feed generation is show in algorithm 2.

Algorithm 2 Algorithm for Pull

Input: *e (Event), User*
Output: *Feed*

```
1:  onEvent(e, User):
2:      User.activity.push(e)
3:
4:  fetchFeed(User):
5:      Feed = []
6:      for each following of User:
7:          Feed.append(following.activity.top)
8:      return Feed
```

Each event by a producer is stored in the activity table of that user. When a feed is requested by a user, the algorithm fetches the top ‘x’ entries of all the producers the user is subscribed to. The value of ‘x’ determines the diversity level of the feed, to ensure items in the feed include events from multiple sources.

In the proposed implementation the user base is divided into a push-list and a pull-list. The deciding factor for this division is the frequency at which each user queries for the feed. This frequency is compared with a global average. If the requesting user's query frequency is lesser, then the user is categorized as a pull user, else the user becomes a push user. The hybrid feed generation algorithm implementation is shown in algorithm 3.

Algorithm 3 Hybrid algorithm for feed generation

Input: *e (Event), User*
Output: *User Feed*

```
 1:  on Event(e, User):
 2:      for each PUSH follower of User
 3:          follower.feed.push(e)
 4:
 5:  fetchFeed(User):
 6:      if User is PUSH:
 7:          return User.Feed
 8:      else if User is PULL
 9:          User.Feed = []
10:          for each following of User:
11:              User.Feed.append(following.activity.top)
12:      return User.Feed
```

In the hybrid algorithm, when an event is created it is still pushed to the subscribers of that user. But here, the push action is performed only to the subscribers categorized as PUSH-target. When a user requests for a feed, there are two ways it could be fetched depending on the category of the user:

- The user is a PUSH-target: In this case all the required events are present right in the feed table of the user. They are ranked, if required and presented to the user.
- The user is a PULL-target: In this case, events need to be fetched from the producers the user is subscribed to. This happens in the same way as the pull model.

CHAPTER 7
SOURCE CODE

CHAPTER 7

SOURCE CODE

7.1 BACK-END (DJANGO)

7.1.1 social_utils.py (Django Backend API)

```
# Provide social functionality to web app and api
# Includes Pull model, Push model, Hybrid model.
# upvote, comment, share, follow

from django.db.models import Sum
from django.shortcuts import get_object_or_404
from rest_framework import status

from website.models import Note, User, Activity


def get_user_pull_feed(auth_user_id):
    user = get_object_or_404(User, auth_user_id=auth_user_id)
    follow_users = Activity.objects.filter(user=user,
activity_type=Activity.FOLLOWS) \
        .values('follows_user__auth_user')
    feed = Activity.objects.filter(user__auth_user__in=follow_users)
    return feed


def get_user_push_feed(auth_user_id):
    user = get_object_or_404(User, auth_user_id=auth_user_id)
    if user.feed_list:
        post_ids = "[{0}]".format(user.feed_list)
        feed = Activity.objects.filter(pk__in=post_ids)
    else:
        feed = Activity.objects.none()
    return feed


def get_user_hybrid_feed(auth_user_id):
    user = get_object_or_404(User, auth_user_id=auth_user_id)

    global_activity_freq =
User.objects.aggregate(Sum('activity_frequency'))
    total_user_count = User.objects.count()
    threshold = global_activity_freq / total_user_count

    if user.activity_frequency >= threshold:
        user.user_type = User.PULL
    else:
        user.user_type = User.PUSH
    user.save(update_fields=['user_type'])

    if user.user_type == User.PULL:
        follow_users = Activity.objects.filter(user=user,
activity_type=Activity.FOLLOWS) \
            .values('follows_user__auth_user')
        feed =
Activity.objects.filter(user__auth_user__in=follow_users)
    else:
        if user.feed_list:
            post_ids = "[{0}]".format(user.feed_list)
            feed = Activity.objects.filter(pk__in=post_ids)
        else:
            feed = Activity.objects.none()
    return feed


def upvote(auth_user_pk, note_pk):
    user = get_object_or_404(User, auth_user_id=auth_user_pk)
    note = get_object_or_404(Note, pk=note_pk)

    try:
        Activity.objects.get(activity_type=Activity.UPVOTE,
user=user, note=note)
        code = status.HTTP_405_METHOD_NOT_ALLOWED
        message = 'You have already upvoted the particular note!'
    except Activity.DoesNotExist:
        Activity.objects.create(activity_type=Activity.UPVOTE,
user=user, note=note)
        code = status.HTTP_200_OK
        message = 'Upvoted successfully!'

    response = {
        'code': code,
        'message': message
    }
    return response


def comment(auth_user_pk, note_pk, comment_text):
    user = get_object_or_404(User, auth_user_id=auth_user_pk)
    note = get_object_or_404(Note, pk=note_pk)

    Activity.objects.create(activity_type=Activity.COMMENT,
user=user, note=note, caption=comment_text)
```

```
    response = {
        'code': status.HTTP_201_CREATED,
        'message': 'Commented successfully!'
    }
    return response


def share(auth_user_pk, note_pk, share_text):
    user = get_object_or_404(User, auth_user_id=auth_user_pk)
    note = get_object_or_404(Note, pk=note_pk)

    try:
        Activity.objects.get(activity_type=Activity.SHARE,
user=user, note=note)
        code = status.HTTP_405_METHOD_NOT_ALLOWED
        message = 'You have already shared the particular note!'
    except Activity.DoesNotExist:
        Activity.objects.create(activity_type=Activity.SHARE,
user=user, note=note, caption=share_text)
        code = status.HTTP_200_OK
        message = 'Shared successfully!'

    response = {
        'code': code,
        'message': message
    }
    return response


def follow(auth_user_pk, follows_user_pk):
    user = get_object_or_404(User, auth_user_id=auth_user_pk)
    follows_user = get_object_or_404(User, pk=follows_user_pk)

    if user == follows_user:
        code = status.HTTP_400_BAD_REQUEST
        message = 'User can not follow themselves!'
        response = {
            'code': code,
            'message': message
        }
        return response

    try:
        Activity.objects.get(activity_type=Activity.FOLLOWS,
user=user, follows_user=follows_user)
        code = status.HTTP_405_METHOD_NOT_ALLOWED
        message = 'You already follow ' +
follows_user.auth_user.username + '!'
    except Activity.DoesNotExist:
        Activity.objects.create(activity_type=Activity.FOLLOWS,
user=user, follows_user=follows_user)
        code = status.HTTP_200_OK
        message = 'Followed successfully!'

    response = {
        'code': code,
        'message': message
    }
    return response


def get_user_activities(auth_user_id):
    user = get_object_or_404(User, auth_user_id=auth_user_id)
    activity_list = Activity.objects.filter(user=user)
    return activity_list


def get_user_posts(auth_user_id):
    user = get_object_or_404(User, auth_user_id=auth_user_id)
    post_list = Note.objects.filter(user=user)
    return post_list


def get_user_followers(user):
    follow_users_ids = Activity.objects.values_list('follows_user',
flat=True)\
        .filter(user=user, activity_type=Activity.FOLLOWS)
    follow_users = User.objects.filter(pk__in=set(follow_users_ids))
    return follow_users


def get_note_comments(note_id):
    note = get_object_or_404(Note, pk=note_id)
    comment_activity_list =
Activity.objects.filter(activity_type=Activity.COMMENT, note=note)
    return comment_activity_list
```

7.1.2 views.py (API views)

```
import hashlib
import json

from django.conf import settings
from django.contrib.auth import login, logout
from django.core.exceptions import ObjectDoesNotExist
from django.http import HttpResponse, HttpResponseBadRequest
from django.shortcuts import get_object_or_404
from django.utils.crypto import get_random_string
from oauth2client.client import OAuth2WebServerFlow
from oauth2client.contrib import xsrfutil
from oauth2client.contrib.django_util.storage import
DjangoORMStorage as Storage
from rest_framework import status
from rest_framework.authentication import SessionAuthentication
from rest_framework.decorators import api_view,
authentication_classes, permission_classes
from rest_framework.generics import ListAPIView,
RetrieveUpdateDestroyAPIView
from rest_framework.parsers import MultiPartParser, FormParser
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.views import APIView

from api.serializers import NoteSerializer, UserCommonSerializer,
ActivitySerializer, UserFullSerializer, \
    NotePublicSerializer
from website.models import Credentials, AuthUser, FbCred
from website.models import Note, User
from website.utils.core_utils import activity_direct_search, upload,
update, delete, download
from website.utils.social_utils import upvote, comment, share,
follow, get_user_activities, \
    get_user_posts, get_user_followers, get_user_pull_feed,
get_user_push_feed, get_user_hybrid_feed, \
    get_note_comments


# ===================================
# ========= User APIs ===============
# ===================================

def handle_exception(exception):
    content = {
        'message': str(exception)
    }
    response = HttpResponse(content=json.dumps(content),
status=status.HTTP_500_INTERNAL_SERVER_ERROR)
    return response


# lists all notes (only GET allowed)
class NoteList(ListAPIView):
    queryset = Note.objects.all()
    serializer_class = NotePublicSerializer


class NoteDetail(RetrieveUpdateDestroyAPIView):
    queryset = Note.objects.all()
    serializer_class = NotePublicSerializer
```

```
class NoteDownload(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request, note_id):
        try:
            auth_user_pk = request.user.pk
            response = download(auth_user_pk, note_id)
            if response.get('code') == status.HTTP_200_OK:
                final_response =
HttpResponse(content=response.get('content'),

content_type=response.get('file_mime_type'))
                final_response['Content-Disposition'] = 'attachment;
filename=' + response.get('file_name')
                return final_response
            else:
                content = {
                    'message': response.get('message')
                }
                final_response =
HttpResponse(content=json.dumps(content),
status=response.get('code'))
                return final_response
        except Exception as e:
            return handle_exception(exception=e)


class NoteComments(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request, note_id):
        try:
            comment_activity_list = get_note_comments(note_id)
            serializer = ActivitySerializer(comment_activity_list,
many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


class ActivitySearch(APIView):

    # Checks if there is a search parameter and filters accordingly
    def get(self):
        try:
            search_term = self.request.query_params.get('q', None)
            if search_term is not None:
                queryset = activity_direct_search(search_term)
                serializer = ActivitySerializer(queryset, many=True)
                return Response(serializer.data)
            else:
                content = {
                    'message': "No search term."
                }
                return HttpResponse(content=json.dumps(content),
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return handle_exception(exception=e)


class UserDetail(APIView):
    def get(self, request, user_pk=None):
        try:
            if user_pk is None:
                auth_user_pk = request.user.pk
                user = get_object_or_404(User,
auth_user_id=auth_user_pk)
            else:
                user = get_object_or_404(User, pk=user_pk)
            serializer = UserFullSerializer(user)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


# Check comments in urls.py for explanation
class UserAction(APIView):
    serializer_class = UserCommonSerializer
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def post(self, request, action_desc):
        try:
            auth_user_pk = request.user.pk
            response = None

            if action_desc == 'upvote':
                note_pk = request.POST.get('note_pk')
                if note_pk:
                    response = upvote(auth_user_pk, note_pk)

            elif action_desc == 'comment':
                note_pk = request.POST.get('note_pk')
                comment_text = request.POST.get('comment_text')
                if note_pk and comment_text:
                    response = comment(auth_user_pk, note_pk,
comment_text)

            elif action_desc == 'share':
                note_pk = request.POST.get('note_pk')
                share_text = request.POST.get('share_text')
                if note_pk and share_text:
                    response = share(auth_user_pk, note_pk,
share_text)

            elif action_desc == 'follow':
                follows_user_pk =
request.POST.get('follows_user_pk')
                if follows_user_pk:
                    response = follow(auth_user_pk, follows_user_pk)

            if response is None:
                response = {
                    'code': status.HTTP_400_BAD_REQUEST,
                    'message': 'Bad request. No operation performed.
Please provide all the params.'
                }
            content = {
                'message': response['message']
            }
            return HttpResponse(content=json.dumps(content),
status=response['code'])
        except Exception as e:
            return handle_exception(exception=e)


class UserActivity(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request, user_pk=None):
        try:
            if user_pk is None:
                auth_user_pk = request.user.pk
            else:
                user = get_object_or_404(User, pk=user_pk)
                auth_user_pk = user.auth_user.pk
            activity_list =
get_user_activities(auth_user_id=auth_user_pk)
            serializer = ActivitySerializer(activity_list,
many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)
```

```
class PublicUserActivity(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request, username=None):
        try:
            if username is None:
                auth_username = request.user.name
            else:
                user = get_object_or_404(User, name=username)
                auth_username = user.auth_user.name
            activity_list =
get_user_activities(auth_user_name=auth_username)
            serializer = ActivitySerializer(activity_list,
many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


# View all posts of the given user
class AllUserPostsList(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request):
        try:
            auth_user_pk = request.user.pk
            post_list = get_user_posts(auth_user_pk)
            serializer = NoteSerializer(post_list, many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


# Let the given user upload, update and delete posts
class UserPost(APIView):
    parser_classes = (MultiPartParser, FormParser)
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    # get user post
    def get(self, request):
        try:
            note_pk = request.GET.get('note_pk')
            if note_pk:
                note = get_object_or_404(Note, pk=note_pk)
                serializer = NoteSerializer(note)
                return Response(serializer.data)
            else:
                content = {
                    'message': "note_pk is required field."
                }
                return HttpResponse(content=json.dumps(content),

status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return handle_exception(exception=e)

    # upload or update user post
    def post(self, request):
        try:
            auth_user_pk = request.user.pk
            note_pk = request.POST.get('note_pk')
            if note_pk:
                response = update(request, note_pk, auth_user_pk)
            else:
                response = upload(request, auth_user_pk)
            content = {
                'message': response['message']
            }
            return HttpResponse(content=json.dumps(content),
                                status=response['code'])
        except Exception as e:
            return handle_exception(exception=e)

    # delete user post
    def delete(self, request):
        try:
            auth_user_pk = request.user.pk
            response = delete(request, auth_user_pk)
            content = {
                'message': response['message']
            }
            return HttpResponse(content=json.dumps(content),
                                status=response['code'])
        except Exception as e:
            return handle_exception(exception=e)


class UserFollowers(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request, user_pk):
        try:
            if user_pk == 'current':
                auth_user_pk = request.user.pk
                user = get_object_or_404(User,
auth_user_id=auth_user_pk)
            else:
                user = get_object_or_404(User, pk=user_pk)
            follower_list = get_user_followers(user)
            serializer = UserCommonSerializer(follower_list,
many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


class UserPullFeed(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request):
        try:
            auth_user_pk = request.user.pk
            feed_list =
get_user_pull_feed(auth_user_id=auth_user_pk)
            serializer = ActivitySerializer(feed_list, many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


class UserPushFeed(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request):
        try:
            auth_user_pk = request.user.pk
            feed_list =
get_user_push_feed(auth_user_id=auth_user_pk)
            serializer = ActivitySerializer(feed_list, many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


class UserHybridFeed(APIView):
    authentication_classes = (SessionAuthentication,)
    permission_classes = (IsAuthenticated,)

    def get(self, request):
        try:
            auth_user_pk = request.user.pk
            feed_list = 
get_user_hybrid_feed(auth_user_id=auth_user_pk)
            serializer = ActivitySerializer(feed_list, many=True)
            return Response(serializer.data)
        except Exception as e:
            return handle_exception(exception=e)


# ===================================
# ======= End of User APIs ==========
# ===================================


# ===================================
# ========= Login APIs ==============
# ===================================

def random_pass():
    allowed_chars = 
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$'
    return get_random_string(16, allowed_chars)


# ===================================
# ======= Helper functions ==========
# ===================================

def new_flow(request):
    flow = OAuth2WebServerFlow(
        client_id=settings.GOOGLE_OAUTH2_CLIENT_ID,
        client_secret=settings.GOOGLE_OAUTH2_CLIENT_SECRET,
        scope=settings.GOOGLE_SCOPE,
        redirect_uri=settings.OAUTH_REDIRECT_URI,
        access_type='offline',
        state='',
        prompt='select_account'
    )
    # check 
https://developers.google.com/identity/protocols/OAuth2WebServer#off
line
    return flow


def gen_username_mail(name, email):
    username = name.replace(" ", "").lower()
    username = username + "-" + str(hashlib.md5(bytes(username + 
email.strip().rsplit('@', 1)[0],

'utf_8')).hexdigest()[:8])
    return username
```

```
def create_user(credentials):
    email = credentials.id_token.get("email")
    name = credentials.id_token.get("name")
    str_picture = credentials.id_token.get("picture")
    picture = str_picture.replace("s96-c","s300-c")
    auth_user = AuthUser.objects.create_user(gen_username_mail(name,
email),
                                             email, random_pass(),
user_type=AuthUser.TYPE_GOOGLE,
                                             name=name,
profile_picture=picture)
    return auth_user

@api_view(['POST'])
def google_login(request):

    if not request.user.is_authenticated():
        flow = new_flow(request)
        flow.params['state'] =
xsrfutil.generate_token(settings.SECRET_KEY,
                                                      request.user)
        # request.session['flow'] = pickle.dumps(flow).decode('iso-
8859-1')
        redirect_uri = flow.step1_get_authorize_url()
        response = {
            'redirect_url': redirect_uri
        }
        return HttpResponse(content=json.dumps(response),
status=status.HTTP_200_OK)
    else:
        response = {
            'message': 'User already logged in!'
        }
        return HttpResponse(content=json.dumps(response),
status=status.HTTP_405_METHOD_NOT_ALLOWED)


@api_view(['POST'])
def google_verify(request):

    # Make sure google sent the request
    if request.data.get('state'):
        if not xsrfutil.validate_token(settings.SECRET_KEY,

request.data.get('state').encode('utf8'),
                                       request.user):
            response = {
                'message': "Cannot verify if Google sent the
request."
            }
            return
HttpResponseBadRequest(content=json.dumps(response))
    else:
        response = {
            'message': "State variable missing."
        }
        return HttpResponseBadRequest(content=json.dumps(response))

    code = request.data.get('code')
    error = request.data.get('error')

    if code and error is None:
        try:
            flow = new_flow(request)
            credentials = flow.step2_exchange(code)
            # not sure if credentials should stored in session
            cred = json.loads(credentials.to_json())
            request.session['creds'] = str(cred['token_response'])
            email = credentials.id_token.get("email")
            user_exists = False
            try:
                user = AuthUser.objects.get(email=email)
                user_exists = True
                str_picture = credentials.id_token.get("picture")
                picture = str_picture.replace("s96-c","s300-c")
                user.profile_picture = picture
            except AuthUser.DoesNotExist:
                user = create_user(credentials)
            if user_exists and user.user_type == AuthUser.TYPE_FB:
                # user has previously logged in through fb , merge
profiles here
                pass
            # Since we've oauth2'd the user, we should set the
backend appropriately
            # This is usually done by the authenticate() method.
            user.backend =
'django.contrib.auth.backends.ModelBackend'

            # Refresh token is needed for renewing google api access
token
            if credentials.refresh_token:
                user.refresh_token = credentials.refresh_token
            user.save()
            login(request, user)
            storage = Storage(Credentials, 'id', request.user,
'credential')
```

```
            storage.put(credentials)

            message = 'Log in successful'
            code = status.HTTP_200_OK

        except Exception as e:
            message = str(e)
            code = status.HTTP_405_METHOD_NOT_ALLOWED

    elif code is None and error:
        message = str(error)
        code = status.HTTP_404_NOT_FOUND

    else:
        message = 'Log in unsuccessful!'
        code = status.HTTP_404_NOT_FOUND

    content = {
        'message': message
    }

    return HttpResponse(content=json.dumps(content), status=code)


@api_view(['POST'])
@authentication_classes((SessionAuthentication,))
@permission_classes((IsAuthenticated,))
def auth_logout(request):
    user = request.user
    # credentials = Credentials.objects.get(id=user.id)
    logout(request)
    # credentials.credential.revoke(httplib2.Http())
    # credentials.delete()
    if user.user_type == AuthUser.TYPE_GOOGLE:
        storage = Storage(Credentials, 'id', user, 'credential')
        storage.delete()
    elif user.user_type == AuthUser.TYPE_FB:
        try:
            storage = FbCred.objects.get(user=user)
            storage.delete()
        except ObjectDoesNotExist as e:
            pass
    message = 'Log out successful!'
    code = status.HTTP_200_OK
    content = {
        'message': message
    }
    return HttpResponse(content=json.dumps(content), status=code)
```

7.2 FRONT-END (REACTJS)

7.2.1 Feed.js (ReactJS news feed component)

```
import React, { Component } from 'react';
import { Col } from 'react-bootstrap';
import Masonry from 'react-masonry-component';

import Card from './Card.js';

import './styles/Feed.css';

var masonryOptions = {
  transitionDuration: 0
};

class Feed extends Component {
  render() {
    const feedItems = this.props.feedItems.map(function(item) {
      return (
          //TODO: Each child in an array or iterator should have a
unique "key" prop.

          <Col xs={12} md={6} className='Feed-item'>
            <div >
              <Card
                userName={item.user.auth_user.username}
                displayName={item.user.auth_user.name}
                activityType={item.activity_type}
                thumbnail={item.note?item.note.thumbnail:''}
                shareCaption={item.caption}
                content={item.note?item.note.post_description:''}

original_poster={item.note?item.note.user_info?item.note.user_info.n
ame:'':''}

op_id={item.note?item.note.user_info?item.note.user_info.username:''
:''}
                notePk={item.note?item.note.id:''}
                numDownloads={item.note?item.note.num_downloads:''}
                numUpvotes={item.note?item.note.num_upvotes:''}
                numComments={item.note?item.note.num_comments:''}
                numShares={item.note?item.note.num_shares:''}

followee={item.follows_user?item.follows_user.auth_user.name:''}

followeeName={item.follows_user?item.follows_user.auth_user.username
:''}
followeePic={item.follows_user?item.follows_user.auth_user.profile_p
icture:''}
              />
            </div>
          </Col>

       );
    });


      return (
       <div className='Feed' >
       <Masonry
         className={'nopadding'} // default ''
         elementType={'ul'} // default 'div'
         options={masonryOptions} // default {}
         disableImagesLoaded={false} // default false
         updateOnEachImageLoad={false} // default false and works
only if disableImagesLoaded is false
       >
         {feedItems}
       </Masonry>
       </div>
    );
  }
}

export default Feed;
```

7.2.5 App.js (Main front-end ReactJS driver code)

```
import React, { Component } from 'react';
import { Col } from 'react-bootstrap';
import { BrowserRouter, Route, Switch, Redirect } from 'react-
router-dom';
import queryString from 'query-string';
import Landing from './components/Landing.js';
import Feed from './components/Feed.js';
import Navbar from './components/Navbar.js';
import Profile from './components/Profile.js';
import Upload from './components/Upload.js';
import Search from './components/Search.js';
import Cookies from 'universal-cookie';
import './App.css';

let CSRF_TOKEN = '';
//Named so to not be tempting to modify value manually
const LOCAL_LOGIN_STATUS = 'enable_js';
```

```
class App extends Component {
  constructor(props) {
    super(props);

    //LocalStorage saves only as strings :/
    const login_status = (localStorage.getItem(LOCAL_LOGIN_STATUS)
=== 'true' ? true : false);
    this.state = {
      is_logged_in: login_status,
      username: '',
      user_details:
{auth_user:{name:''},num_posts:0,num_followers:0,num_following:0},
      feed_items: [
        {id:1, activity_type:"UPLOAD", caption: 'This be caption for
this note file. Lorem Ipsum Something.',
        user:{auth_user:{username:'smallbruce', name:'WUCE Brain'}},
note:{num_comments:12, num_downloads:23, num_shares:34,
num_upvotes:45}},
        {id:1, activity_type:"UPLOAD", caption: 'This be caption for
this note file. Lorem Ipsum Something.',
        user:{auth_user:{username:'smallbruce', name:'WUCE Brain'}},
note:{num_comments:12, num_downloads:23, num_shares:34,
num_upvotes:45}},
        {id:1, activity_type:"UPLOAD", caption: 'This be caption for
this note file. Lorem Ipsum Something.',
        user:{auth_user:{username:'smallbruce', name:'WUCE Brain'}},
note:{num_comments:12, num_downloads:23, num_shares:34,
num_upvotes:45}},
        {id:1, activity_type:"UPLOAD", caption: 'This be caption for
this note file. Lorem Ipsum Something.',
        user:{auth_user:{username:'smallbruce', name:'WUCE Brain'}},
note:{num_comments:12, num_downloads:23, num_shares:34,
num_upvotes:45}}
      ]
    };
    this.state.search_items = this.state.feed_items;
  }

  componentDidMount() {
    this.feed_me();
    this.user();
  }

  update_csrf_token() {
    const cookies = new Cookies();
    CSRF_TOKEN = cookies.get('csrftoken');
  }
```

```
user() {
  this.update_csrf_token();
  fetch('http://localhost:8000/api/user/', {
      method: 'GET',
      headers: {
        'Accept': 'application/json',
        'Content-type': 'application/json',
        'X-CSRFToken': CSRF_TOKEN
      },
      credentials: 'include'
    }
  )
  .then(response => {
    console.log('User api response: ', response);
    if(response.status !== 200)
      throw Error(response.status);
    return response.json();
  })
  .then(json => {
    console.log('User api response json: ', json);
    this.setState({user_details: json})
  })
  .catch(error => {
    console.log('user api error: ', error);
  });
}

feed_me() {
  this.update_csrf_token();
  fetch('http://localhost:8000/api/user/pull_feed/', {
      method: 'GET',
      headers: {
        'Accept': 'application/json',
        'Content-type': 'application/json',
        'X-CSRFToken': CSRF_TOKEN
      },
      credentials: 'include'
    }
  )
  .then(response => {
    console.log('feed api response: ', response);
    if(response.status !== 200)
      throw Error(response.status);
    return response.json();
  })
  .then(json => {
    console.log('feed api response json: ', json);
    this.setState({feed_items: json})
  })
```

```
    .catch(error => {
      console.log('feed api error: ', error);
    });
  }

  login_verify(params, routeProps) {
    if(!this.state.is_logged_in) {
      const params_body = JSON.stringify({
        state: params.state,
        code: params.code,
        error: params.error
      });

      this.update_csrf_token();
      fetch('http://localhost:8000/api/google_verify/', {
        method: 'POST',
        headers: {
          'Accept': 'application/json',
          'Content-type': 'application/json',
          'X-CSRFToken': CSRF_TOKEN
        },
        body: params_body,
        credentials: 'include'
      })
      .then(response => {
        console.log('google_verify response: ', response);
        if(response.status !== 200)
          throw Error(response.status);
        return response.json();
      })
      .then(json => {
        console.log('google_verify response json: ', json);
        localStorage.setItem(LOCAL_LOGIN_STATUS, 'true');
        this.setState({is_logged_in: true});
        //Fetch feed on login
        this.user();
        this.feed_me();
        routeProps.history.push('/');
      })
      .catch(error => {
        //TODO: Handle login failure
        console.log('google_verify error: ', error);
        localStorage.setItem(LOCAL_LOGIN_STATUS, 'false');
        this.setState({is_logged_in: false});
        routeProps.history.push('/ferr');
      });
    }
  }
  logout() {
    this.update_csrf_token();
    fetch('http://localhost:8000/api/logout/', {
      method: 'POST',
      headers: {
        'Accept': 'application/json',
        'Content-type': 'application/json',
        'X-CSRFToken': CSRF_TOKEN
      },
      credentials: 'include',
    })
    .then(response => {
      console.log('google_logout response: ', response);
      if(response.status !== 200)
        throw Error(response.status);
      return response.json();
    })
    .then(json => {
      console.log('google_logout response json: ', json);
      localStorage.setItem(LOCAL_LOGIN_STATUS, 'false');
      this.setState({is_logged_in: false});
    })
    .catch(error => {
      console.log('logout error: ', error);
    });
  }

  render() {
    return (
      <div className='App'>
      <BrowserRouter>
        <Switch>
          <Route                    /* Home */
            exact path='/'
            render={ routeProps => {
                return(
                  this.state.is_logged_in ?
                  <div>
                    <Navbar {...routeProps}
isLoggedIn={this.state.is_logged_in}
username={this.state.user_details.auth_user.username}
userFullName={this.state.user_details.auth_user.name}/>
                    <Col xs={12} md={10} mdOffset={1}>
                      <Feed feedItems={this.state.feed_items}/>
                    </Col>
                  </div>
                  : <Redirect to='/welcome'/>
                );
              }
            }
          />
          <Route                    /* Landing page */
            exact path='/welcome'
            render={ routeProps =>
              <div>
                <Navbar {...routeProps}
isLoggedIn={this.state.is_logged_in}
username={this.state.user_details.auth_user.name}
userFullName={this.state.user_details.auth_user.name}/>
                <Landing/>
              </div>
            }
          />
          <Route                    /* Upload */
            exact path='/upload'
            render={ routeProps =>
              <div>
                <Navbar {...routeProps}
isLoggedIn={this.state.is_logged_in}
username={this.state.user_details.auth_user.name}
userFullName={this.state.user_details.auth_user.name}/>
                <Upload/>
              </div>
            }
          />
          <Route                    /* Logout */
            exact path='/logout'
            render={ routeProps => {
              this.logout();
              routeProps.history.push('/');
                return(
                  <div>
                    Logging out, please wait...
                  </div>
                );
              }
            }
          />
          <Route
            path='/from_google/'  /* Redirected from google */
            render={ routeProps =>
              {
                if(!this.state.is_logged_in)

this.login_verify(queryString.parse(routeProps.location.search),
routeProps);
                return (
                  <div>
```

```
                    Authenticating, please wait...
                  </div>
                );
              }
            }
          />
          <Route
            exact path='/search/'   /* Search */
            render={ routeProps =>
              <div>
                <Navbar {...routeProps}
isLoggedIn={this.state.is_logged_in}
username={this.state.user_details.auth_user.name}
userFullName={this.state.user_details.auth_user.name}/>
                <Search
searchKey={queryString.parse(routeProps.location.search)['q']}/>
              </div>
            }
          />
          {
          <Route                    /* User profile */
            exact path='/:userFullName'
            render={ routeProps =>
              <div>
                <Navbar {...routeProps}
isLoggedIn={this.state.is_logged_in}
username={this.state.user_details.auth_user.username}
userFullName={this.state.user_details.auth_user.name}/>
                <Profile
username={this.state.user_details.auth_user.username}

userFullName={this.state.user_details.auth_user.name}

profilePic={this.state.user_details.auth_user.profile_picture}
                 numPosts={this.state.user_details.num_posts}

numFollowers={this.state.user_details.num_followers}

numFollowing={this.state.user_details.num_following}/>
              </div>
            }
          />
        </Switch>
      </BrowserRouter>
      </div>
    );
  }
}
export default App;
```

CHAPTER 8
TESTING

CHAPTER 8

TESTING

8.1 INTRODUCTION

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

During the development of software, errors can be injected at any stage. However, requirements and design errors are likely to remain undetected. Those errors will be ultimately reflected in the code. During testing, the program to be tested is executed with a set of test cases and output of program for test cases are evaluated to describe the program performance to expected level. No system design is perfect because of communication problem, programmer‘s negligence or constraints create errors that must be eliminated before the system is ready for use of acceptance.

There are various types of test. Each test type addresses a specific testing requirement. Following sections describes the system testing and test cases.

8.2 UNIT TESTING

This is the first level of testing, where different components are tested against the requirement specification for the individual components. Unit testing is essential for verification of the code produced during the coding phase and hence the goal is to test internal logic of those components. In our project each and every ReactJS component has undergone unit testing for functionality and robustness**.**

8.3 SYSTEM TESTING

System testing is performed on the entire system in the context of functional requirements specification and system requirement specifications. System testing tests not only the design, but also the behavior and the expectations.it is also to test up to and beyond the bounds defined in the software requirement specification.

System testing for our platform is accomplished by the following test cases. These test cases show how each and every component interacts with each other as a dynamic system.

TEST CASE ID	**PAGE**	**TEST DATA**	**EXPECTED RESULT**	**ACTUAL RESULT**	**STATUS**
TC01	Landing	Click Signup/Login	Open Google Auth form	Open Google Auth form	PASS
TC02	Google Auth	Valid uname & pwd	Open user feed	Open user feed	PASS
TC03	Google Auth	Valid uname & Invalid pwd	Show error	Show error	PASS
TC03	User feed	Click Hamburger button	Show menu	Show menu	PASS
TC04	User feed	Click on upvote	Upvote post	Upvote post	PASS
TC05	User feed	Click on share	Share post	Share post	PASS
TC06	User feed	Click on comment	Show comment page	Show comment page	PASS
TC07	User feed	Click on download	Download post	Download post	PASS
TC08	User feed	Click on profile from menu	Show profile page	Show profile page	PASS
TC09	User feed	Click on logout from menu	Show logging out message	Show logging out message	PASS
TC10	User feed	Click on upload from menu	Show upload form	Show upload form	PASS
TC11	Upload	Fill in required fields and click submit	Show successful submission	Show successful submission	PASS
TC12	Upload	Miss out on a few required fields	Show fill required fields message	Show fill required fields message	PASS
TC13	Profile	Click on upvote	Upvote post	Upvote post	PASS
TC14	Profile	Click on share	Share post	Share post	PASS
TC15	Profile	Click on comment	Show comment page	Show comment page	PASS
TC16	Profile	Click on download	Download post	Download post	PASS
TC17	Comment	Submit comment	Post comment	Post comment	PASS

Table 8.1 Quicknotes Platform Test Cases

CHAPTER 9
SNAPSHOTS

CHAPTER 9

SNAPSHOTS

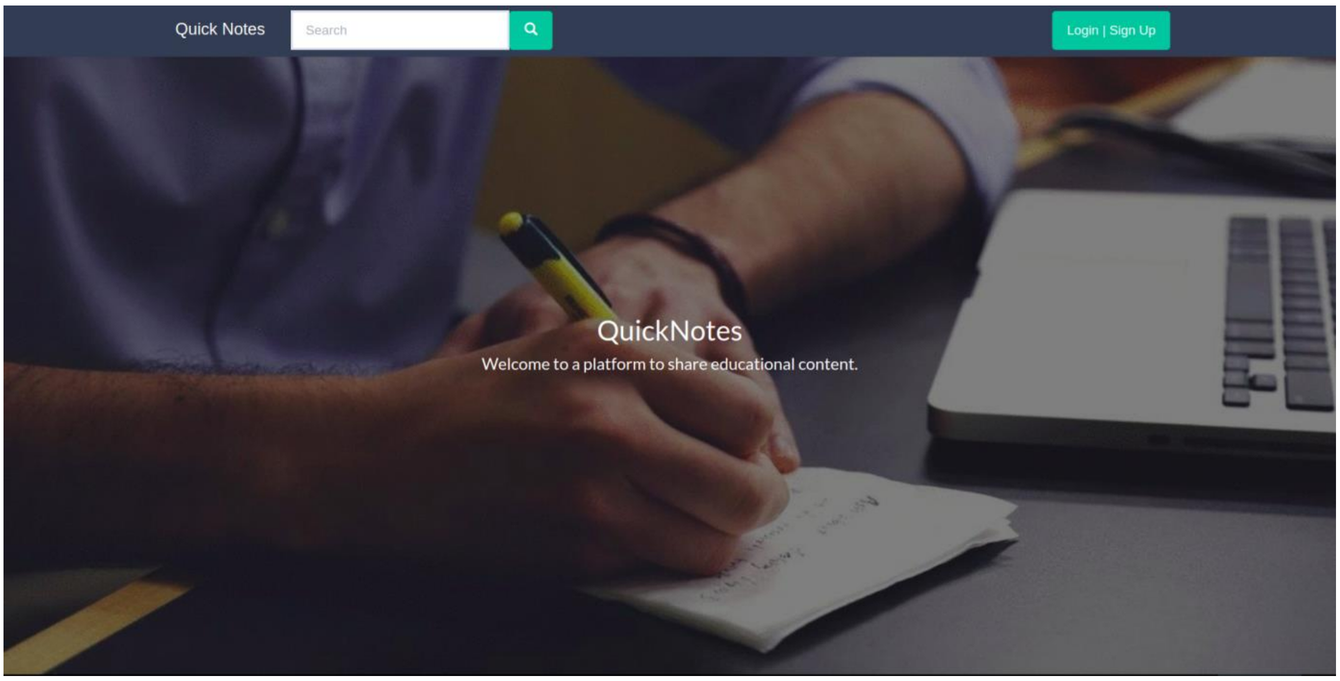


Fig. 9.1 The Landing Page for QuickNotes

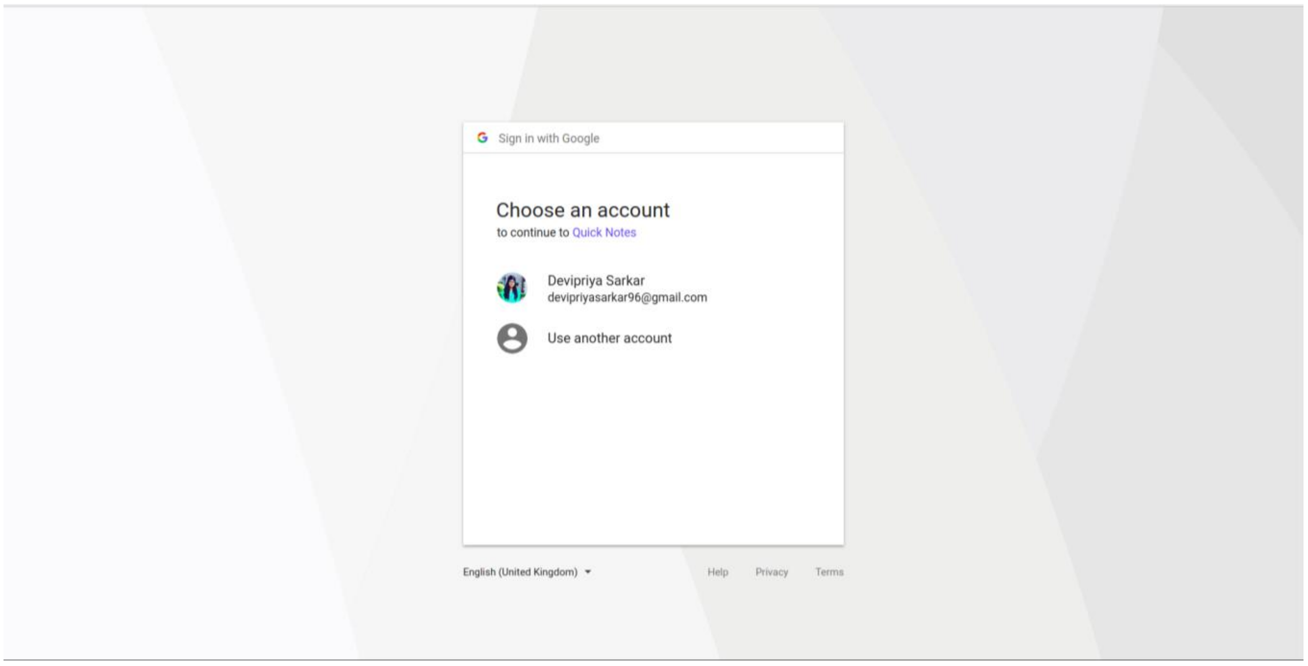


Fig. 9.2 Google Authentication Page

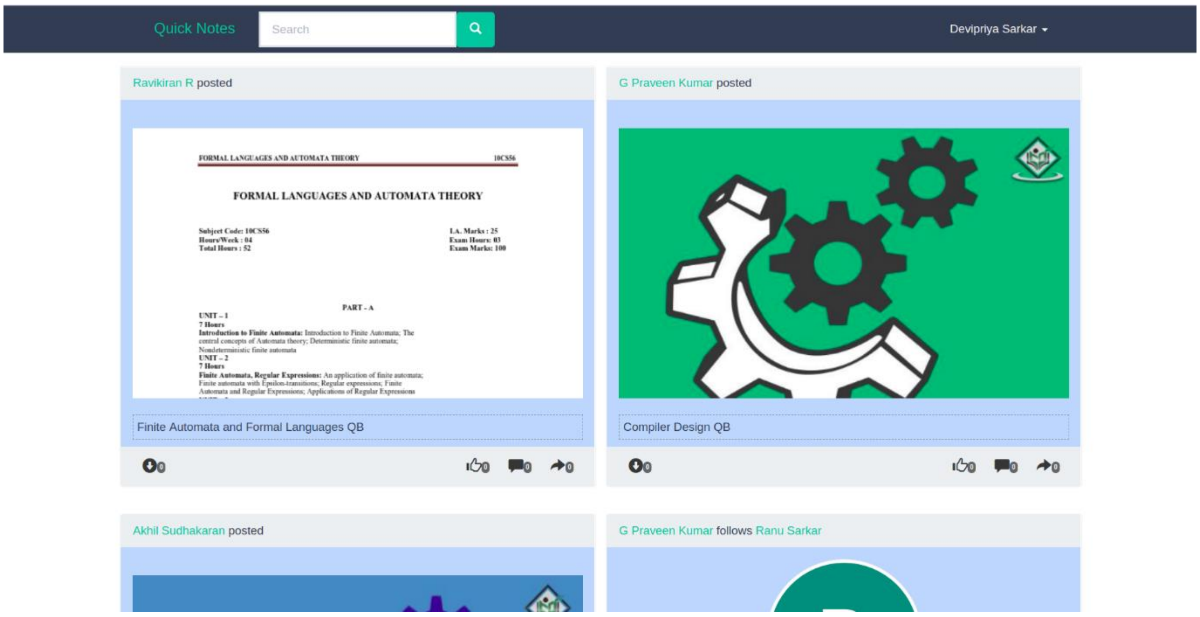
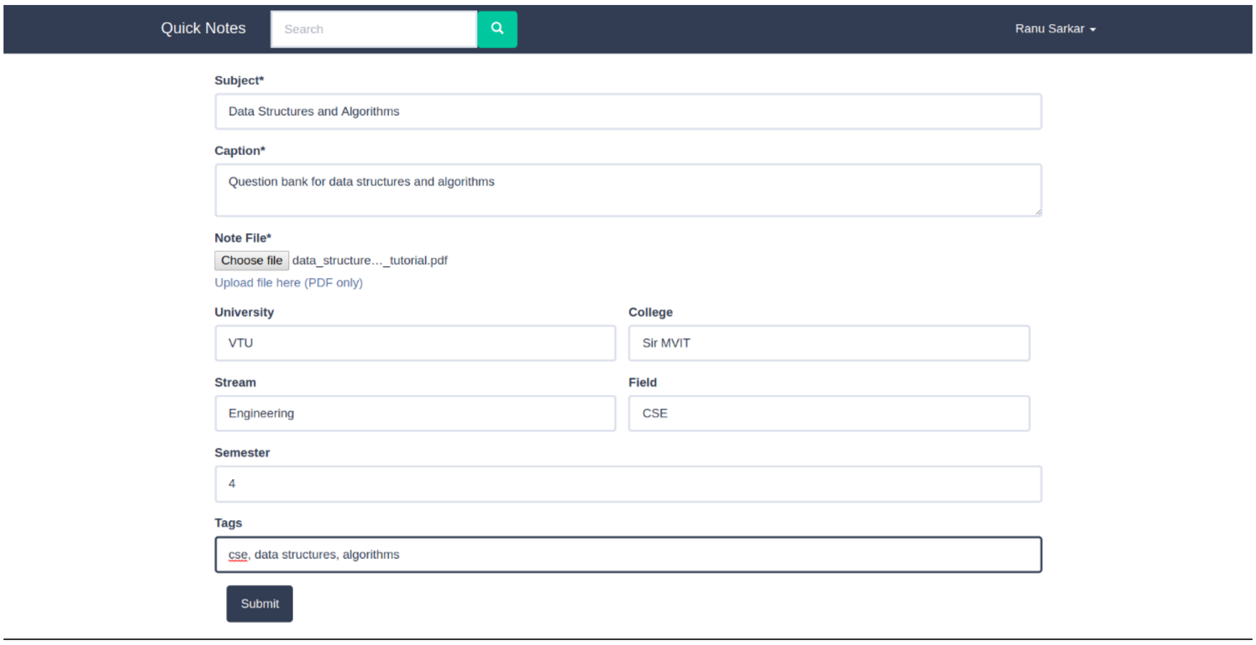


Fig. 9.3 Feed Page



Subject*
Data Structures and Algorithms

Caption*
Question bank for data structures and algorithms

Note File*
data_structure..._tutorial.pdf

University: VTU

College: Sir MVIT

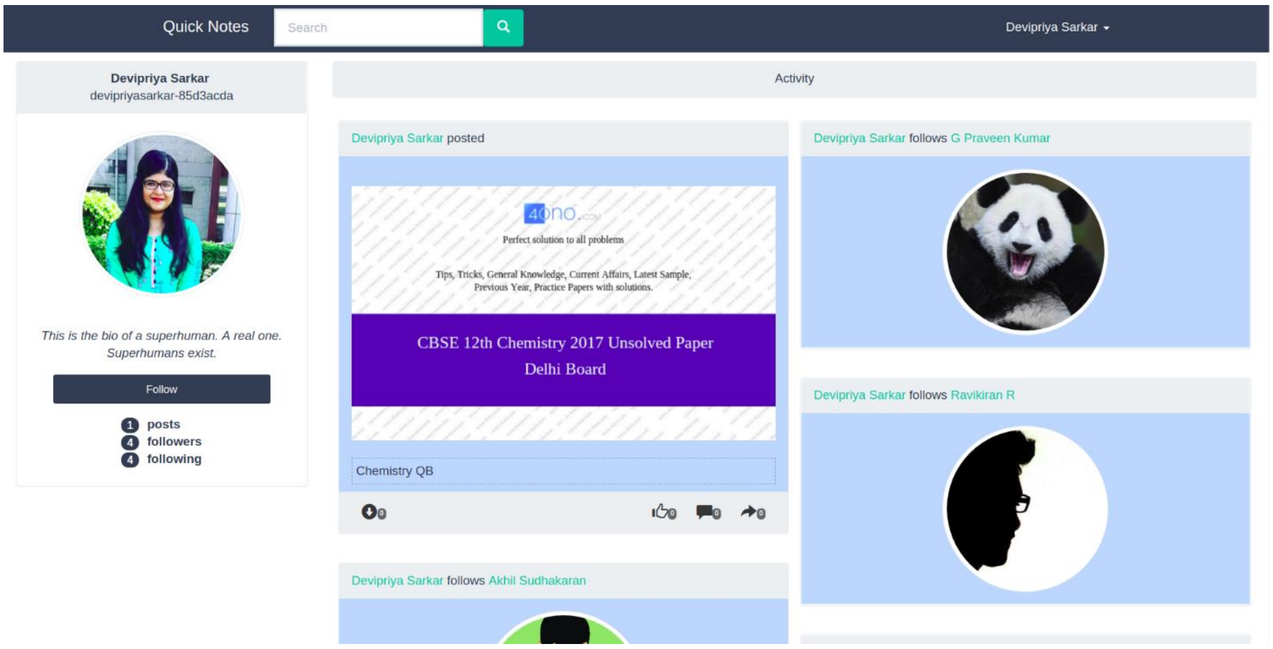
Stream: Engineering

Field: CSE

Semester: 4

Tags: cse, data structures, algorithms

Fig. 9.4 Upload Form



Quick Notes Search Devipriya Sarkar

Devipriya Sarkar
devipriyasarkar-85d3acda

This is the bio of a superhuman. A real one. Superhumans exist.

Follow

1 posts
4 followers
4 following

Activity

Devipriya Sarkar posted

4ono.com
Perfect solution to all problems
Tips, Tricks, General Knowledge, Current Affairs, Latest Sample, Previous Year, Practice Papers with solutions.
CBSE 12th Chemistry 2017 Unsolved Paper
Delhi Board

Chemistry QB

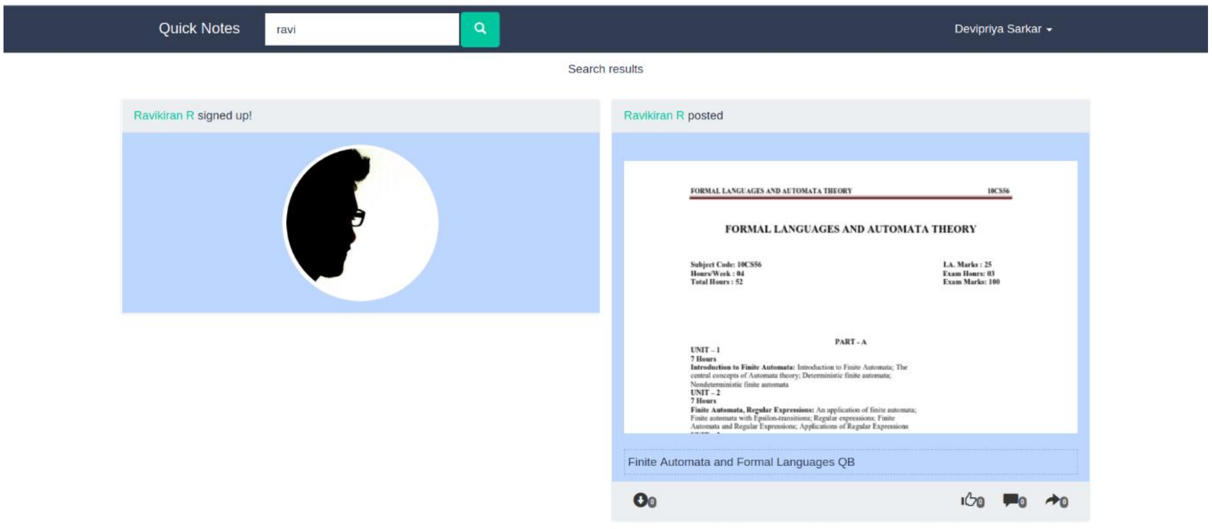
0 0 0 0

Devipriya Sarkar follows G Praveen Kumar

Devipriya Sarkar follows Ravikiran R

Devipriya Sarkar follows Akhil Sudhakaran

Fig. 9.5 Profile Page



Quick Notes ravi Devipriya Sarkar

Search results

Ravikiran R signed up!

Ravikiran R posted

FORMAL LANGUAGES AND AUTOMATA THEORY 10CS56

FORMAL LANGUAGES AND AUTOMATA THEORY

Subject Code: 10CS56
Hours/Week : 04
Total Hours : 52

I.A. Marks : 25
Exam Hours: 03
Exam Marks: 100

PART - A
UNIT – 1
7 Hours
Introduction to Finite Automata: Introduction to Finite Automata; The central concepts of Automata theory; Deterministic finite automata; Nondeterministic finite automata
UNIT – 2
7 Hours
Finite Automata, Regular Expressions: An application of finite automata; Finite automata with Epsilon-transitions; Regular expressions; Finite Automata and Regular Expressions; Applications of Regular Expressions

Finite Automata and Formal Languages QB

0 0 0 0

Fig. 9.6 Search Result

CHAPTER 10
RESULTS

CHAPTER 10

RESULTS

Given below are the comparisons between response times of the different models of feed generation for different number of users.

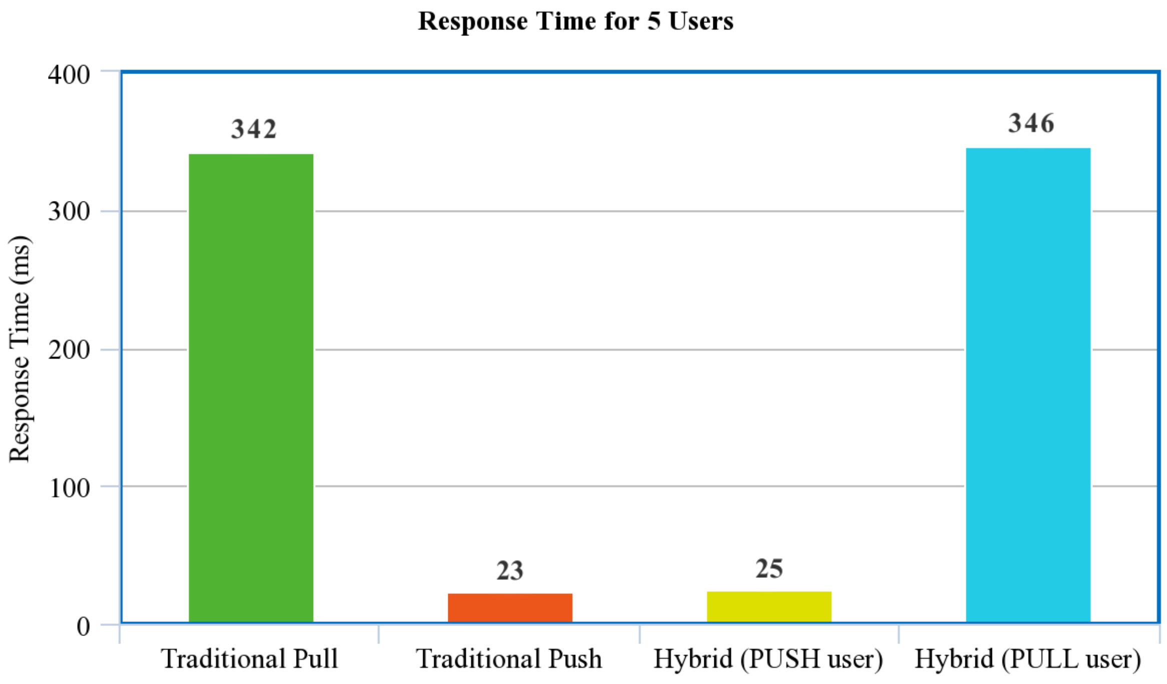


Fig. 10.1 Response time for 5 users

We can see in the above chart that the advntage of using a hybrid model does not immidiately appear in the case of just 5 users. Let us now observe the analytical statistics of a case where the number of users are 10 and 25 as seen in figure 10.2 and 10.3.

As observed from the charts we can see that on increasing the count of user gradually, the decrease in response time happens dramatically for a PUSH user target in a hybrid model setup. This gain can be seen even with even larger user-bases since the backend is developed using Django and it can handle scale well. Since we know for sure that for every very active user there is a user that is less active, it gives rise to the assumption that at any given time there are 50% PUSH target users and 50% PULL target users, and those PUSH users experience a significantly better response time.

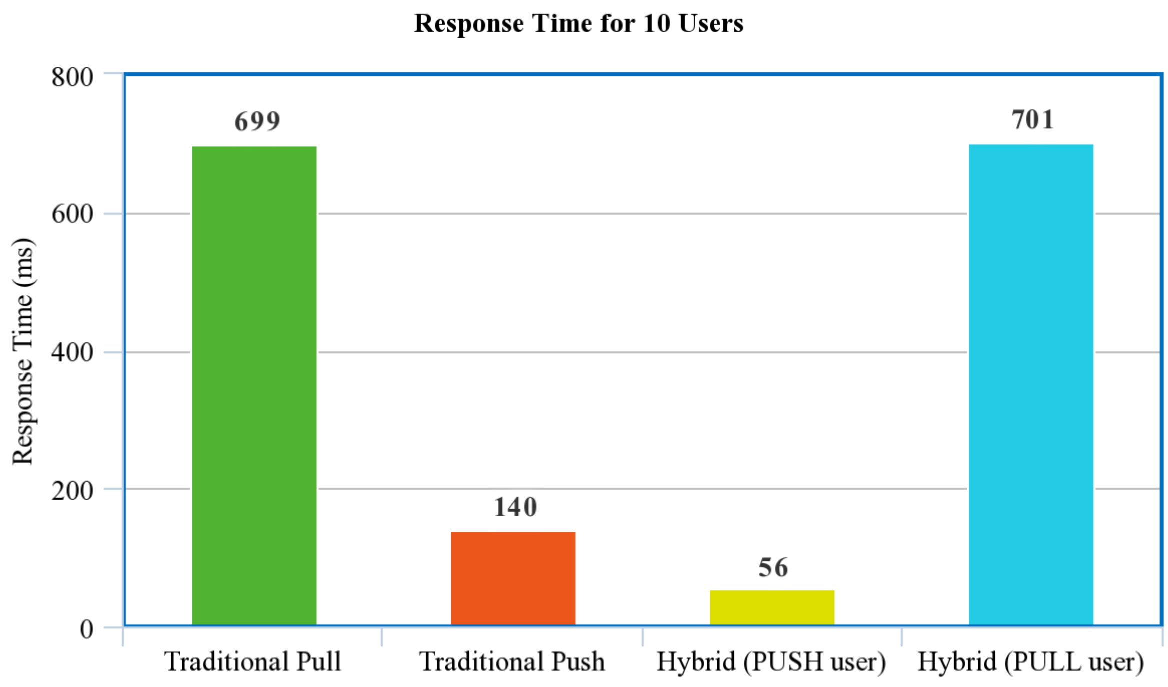


Fig. 10.2 Response time for 10 users

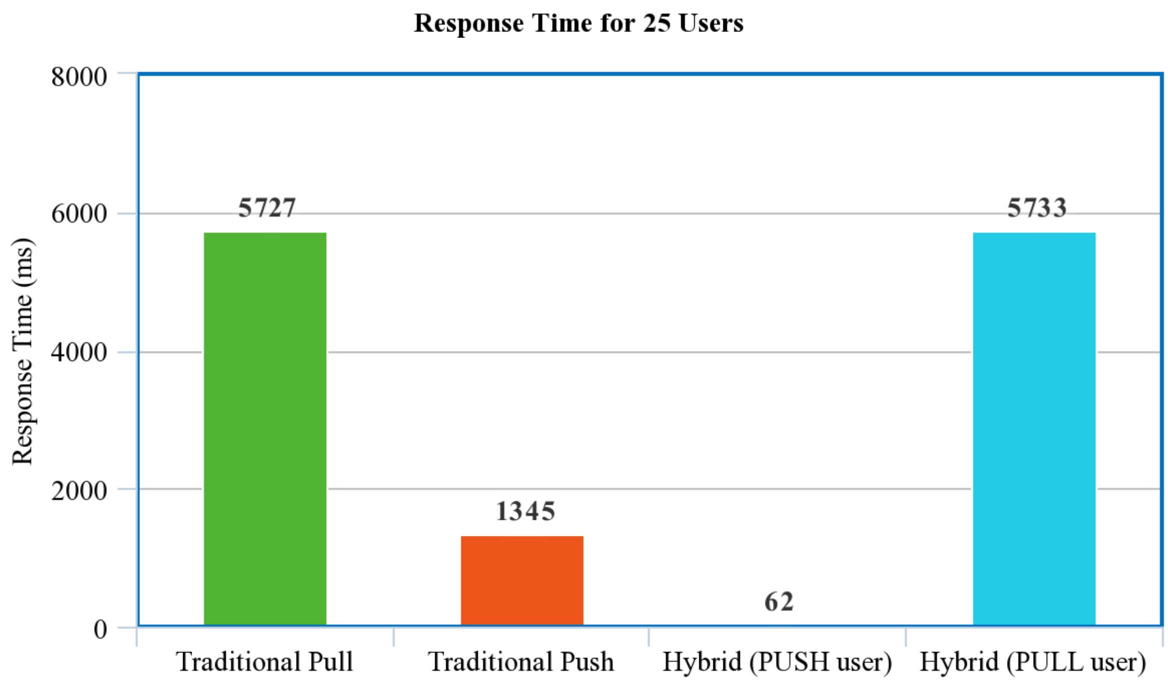


Fig. 10.3 Response time for 25 users

CHAPTER 11
CONCLUSION & FUTURE ENHANCEMENTS

CHAPTER 11

CONCLUSION AND FUTURE WORK

11.1 CONCLUSION

An attempt has been made to develop a hybrid scalable feed generation algorithm which tries to achieve the functionality of traditional feed generation systems and at the same time making its approach simpler and efficient. This project gave us an insight into the various ways in which different social networks or microblogging services handle feed generation.

As seen on the result graphs in the previous chapter, we can conclude that this hybrid model effectively splits the computational load between servers and clients and also brings about a gain in terms of response time during feed fetching in a simpler and elegant method.

11.2 FUTURE WORK

- Implementing techniques to remove seen posts on the feed to limit size after publishing them.
- Implement techniques to improve diversity of posts.
- Experimenting the working of the hybrid model by using a different heuristic than online activity for categorization of users as a PUSH or PULL target.
- To include a recommendation system to make the feed more personal and unique.

11.3 PROJECT ACTIVITY

This project was completed in a total duration of three months (90 days). Monthly project reviews for the same were conducted in the months of March, April and May. The Gantt chart of our project activity is shown below:

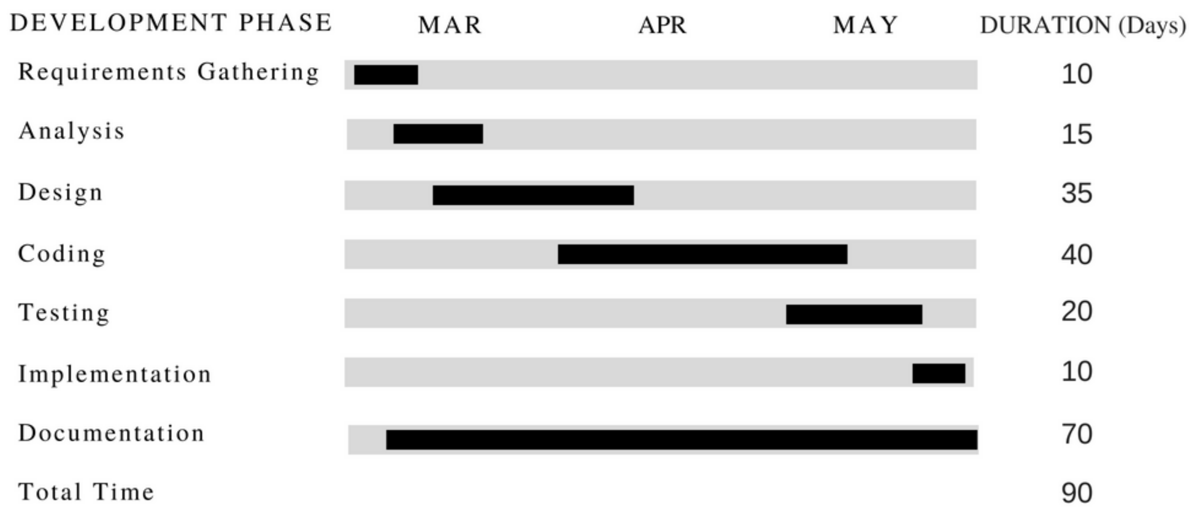


Fig. 11.1 Gantt Chart

REFERENCES

REFERENCES

[1]. https://blog.twitter.com/engineering/en_us/topics/infrastructure/2017/the-infrastructure-behind-twitter-scale.html

[2]. V. Sharma et al. "Scaling Deep Social Feeds at Pinterest" *SocialCom,* 2013.

[3]. https://blogs.vmware.com/vfabric/2013/04/how-instagram-feeds-work-celery-and-rabbitmq.html

[4]. http://highscalability.com/blog/2013/10/28/design-decisions-for-scaling-your-high-traffic-feeds.html

[5]. http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html

[6]. A. Silberstein et al. "Feeding Frenzy: Selectively Materializing Users' Event Feeds" *SIGMOD,* June 6–11, 2010.

[7]. J. Dean. "Designs, lessons and advice from building large distributed systems" *3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware* 2009.

[8]. https://en.wikipedia.org/wiki/Activity_stream

[9]. https://aws.amazon.com/message-queue/

[10]. https://code.facebook.com/posts/781984911887151/serving-facebook-multifeed-efficiency-performance-gains-through-redesign/

[11]. https://engineering.instagram.com/what-powers-instagram-hundreds-of-instances-dozens-of-technologies-adf2e22da2ad

[12]. https://www.quora.com/What-are-the-best-practices-for-building-something-like-a-news-feed

[13]. https://fanout.io/docs/smartfeeds.html

[14]. https://www.slideshare.net/danmckinley/etsy-activity-feeds-architecture

[15]. https://stream-framework.readthedocs.io/en/latest/